**НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ**

**«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»**

**Факультет інформатики та обчислювальної техніки Кафедра інформаційних систем та технологій**

До захисту допущено:

Завідувач кафедри

\_\_\_\_\_\_\_\_\_\_ Олександр РОЛІК

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2025 р.

**Дипломний проєкт**

**на здобуття ступеня бакалавра**

**за освітньо—професійною програмою «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» на тему: «Підсистема автоматизованого керування складськими роботами»**

Виконав:

Студент IV курсу, групи Ік—12

Чубко Михайло Ігорович \_\_\_\_\_\_\_\_\_\_ (підпис)

Керівник:

Асистент кафедри, PhD Головатенко Ілля Анатолійович

\_\_\_\_\_\_\_\_\_\_ (підпис)

Рецензент:

Посада, науковий ступінь …..

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань. Студент \_\_\_\_\_\_\_\_\_\_\_\_\_ (підпис)

Київ – 2025 року

**Національний технічний університет України**

**«Київський політехнічний інститут імені Ігоря Сікорського»**

**Факультет інформатики та обчислювальної техніки**

**Кафедра інформаційних систем та технологій**

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо—професійна програма «Інформаційне забезпечення робототехнічних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

\_\_\_\_\_\_\_ Олександр РОЛІК

«\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_2025 р.

**ЗАВДАННЯ на дипломний проєкт студенту Чубко Михайла Ігоровича**

1. Тема проєкту «Підсистема автоматизованого керування складськими роботами», керівник проєкту Асистент кафедри, PhD Головатенко Ілля Анатолійович, затверджені наказом по університету від «7» травня 2025 р. №\_\_\_\_\_
2. Термін подання студентом проєкту \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Вихідні дані до проєкту: Програмне забезпечення.

1. Зміст роботи: Аналіз сучасних типових складських систем, визначення основних недоліків таких систем, розробка та реалізація програмного забезпечення для автономної складської системи онлайн магазину з використанням мобільних роботів, порівняння та вибір найкращого алгоритму пошуку шляху для обраного середовища.
2. Перелік графічного матеріалу (із зазначенням обов’язкових креслеників, плакатів, презентацій тощо): «Презентаціяв в розробці»

6. Дата видачі завдання 15 серпня 2025 року

# **Календарний план**

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назва етапів виконання дипломного проєкту | Термін виконання етапів проєкту | Примітка |
| 1 | Визначення напрямку роботи, ознайомлення з темою. | 01.15.2025 – 01.30.2025 | Виконано |
| 2 | Постановка завдання та аналіз предметної області | 02.15.2025 – 2.28.2025 | Виконано |
| 3 | Проєктування архітектури майбутнього проєкту: уявне представлення структури підсистеми, вибір технологій, інструментів, ресурсів. | 02.03.2025 – 03.10.2025 | Виконано |
| 4 | Проєктування та розробка БД для підсистеми, підключення та налагодження. | 03.12.2025 – 3.18.2025 | Виконано |
| 5 | Проєктування та реалізація логіки імітаційних модулів, а саме основних бізнес—процесів системи (оформлення, прийом, обробка замовлень тощо) | 03.18.2025 – 03.25.2025 | Виконано |
| 6 | Створення адмін панелі користувача, реалізація мапи складу | 03.25.2025 – 03.27.2025 | Виконано |
| 7 | Інтеграція та розробка алгоритмів пошуку найкоротшого шляху для мобільних роботів, обробка замовлень роботами | 03.27.2025 – 04.20.2025 | Виконано |
| 8 | Розробка алгоритму дій для уникнення колізій, помилок в роботі роботів, тестування | 04.20.2025 – 05.05.2025 | Виконано |
| 9 | Тестування інтегрованих алгоритмів пошуку найкоротшого шляху та опис результатів | 05.05.2025 – 05.10.2025 | Виконано |
| 10 | Написання пояснювальної записки | 05.10.2025 – 06.??.2025 |  |
| 11 | Передзахист дипломної роботи | 06.28.2025 |  |
| 12 | Захист дипломної роботи | 06.??.2025 |  |

|  |  |  |
| --- | --- | --- |
| Студент |  | Чубко Михайло Ігорович |
|  |  |  |
| Керівник проєкту |  | Головатенко Ілля Анатолійович |

Анотація

Чубко М.І. Підсистема автоматизованого керування складськими роботами. Кпі ім. Ігоря Сікорського, Київ, 2025.

Обсяг пояснювальної записки складає 109 сторінок. Кількість використаних джерел 16 та х рисунків.

**Актуальність роботи:**

У сучасних умовах стрімкого розвитку логістики та електронної комерції питання автоматизації складських процесів набуває особливої важливості. Традиційні методи ручного керування складом виявляються недостатньо ефективними в умовах зростаючих обсягів товарообігу, потреби в оперативній обробці замовлень та мінімізації людського фактора. Застосування інтелектуальних систем управління складом дозволяє оптимізувати логістичні потоки, скоротити час обробки замовлень, зменшити кількість помилок і забезпечити постійний контроль за інвентарем у реальному часі з мінімальним втручанням людини або взагалі без неї.

**Об’єкт дослідження:**

Інтелектуальна підсистема управління складом з використанням мобільних роботів для автоматизації обробки замовлень, переміщення товарів та ведення інвентаризації.

**Мета роботи**:  
 Розробка функціональної та візуалізованої інформаційної підсистеми для управління складською логістикою із залученням алгоритмів маршрутного планування та роботизованого переміщення товарів з палет на полиці.

**Методи, що використовуються**:

У межах проєкту було здійснено проєктування та моделювання бази даних на платформі MS SQL Server, а також алгоритмізацію логіки обробки замовлень із використанням мови програмування Python та бібліотеки pyodbc. Для адміністративного керування було реалізовано графічний інтерфейс за допомогою бібліотеки Tkinter. Система підтримує імітаційне моделювання з можливістю оновлення координат об’єктів у реальному часі. Крім того, запроваджено механізм пошуку маршрутів з урахуванням колізій та координатної сітки складу, з використанням алгоритмів A\* та dijkstra’s.

**Отримані результати**:

Створено симулятор складу з графічним відображенням основних елементів полиць, палет, роботів і зарядної станції. Впроваджено повний цикл обробки замовлень від їх генерації до розміщення товарів на відповідних полицях. Автоматизовано оновлення координат роботів із візуалізацією їхнього руху, а також проведенне тестування та порівняння алгоритмів руху.

**Рекомендації до використання**

Система може бути використана як навчальна платформа для моделювання інтелектуальних агентів та логістичних процесів, та як основа для реальних автоматизованих складських рішень малого/середнього масштабу, або для подальших досліджень у сфері робототехніки та інформаційних систем.

**Ключові слова**:

Інформаційна підсистема, склад, робот, автоматизація, маршрут, SQL, Python, Tkinter, візуалізація, логістика, алгоритм планування, симуляція.

SUMMARY

**Relevance of the Study:**

In today's rapidly evolving landscape of logistics and e—commerce, the automation of warehouse processes has become increasingly important. Traditional manual warehouse management methods are proving insufficient in the face of growing product turnover, the need for fast order processing, and the minimization of human error. The use of intelligent warehouse management systems enables the optimization of logistics flows, reduction of order processing time, minimization of mistakes, and real—time inventory control with minimal or no human intervention.

**Object of the Study:**

An intelligent warehouse management subsystem utilizing mobile robots to automate order processing, goods transportation, and inventory management.

**Purpose of the Study:**

To develop a functional and visualized information subsystem for warehouse logistics management, integrating path—planning algorithms and robotic movement for transferring goods from pallets to shelves.

**Methods Used:**

The project involved the design and modeling of a database using MS SQL Server, along with the development of order processing logic implemented in Python using the pyodbc library. A graphical user interface was created for administrative management using the Tkinter library. The system supports simulation modeling with real—time coordinate updates of objects. Additionally, a pathfinding mechanism was implemented, considering collisions and the warehouse’s coordinate grid, using algorithms such as A\* and Dijkstra’s.

**Results Obtained:**

A warehouse simulator was developed, featuring graphical representations of key elements such as shelves, pallets, robots, and a charging station. A full order processing cycle was implemented — from order generation to product placement on appropriate shelves. The robots’ coordinate updates were automated and visualized, and the movement algorithms were tested and compared.

**Recommendations for Use:**

The system can be utilized as an educational platform for modeling intelligent agents and logistics processes, as a basis for real small— or medium—scale automated warehouse solutions, or for further research in the fields of robotics and information systems.

**Keywords:**

Information subsystem, warehouse, robot, automation, route, SQL, Python, Tkinter, visualization, logistics, planning algorithm, simulation.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| № рядка | Формат | Позначення | Найменування | Кіл. аркушів | № екз. | Примітка |
| 1 |  |  | Документація загальна |  |  |  |
| 2 |  |  |  |  |  |  |
| 3 |  |  | Знову розроблена |  |  |  |
| 4 |  |  |  |  |  |  |
| 5 | А4 | ІК12.310БАК.006 ПЗ | Підсистема автоматизованого керування складськими роботами | 109 |  |  |
| 6 |  |  | Пояснювальна записка |  |  |  |
| 7 |  |  |  |  |  |  |
| 8 | А3 | ІК12.310БАК.006 Д1 |  | 1 |  |  |
| 9 |  |  |  |  |  |  |
| 10 |  |  |  |  |  |  |
| 11 | А3 | ІК12.310БАК.006 Д2 |  | 1 |  |  |
| 12 |  |  |  |  |  |  |
| 13 |  |  |  |  |  |  |
| 14 | А3 | ІК12.310БАК.006 Д3 |  | 1 |  |  |
| 15 |  |  |  |  |  |  |
| 16 |  |  |  |  |  |  |
| 17 | А3 | ІК12.310БАК.006 Д4 |  | 1 |  |  |
| 18 |  |  |  |  |  |  |
| 19 | А3 | ІК12.310БАК.006 Д5 |  | 1 |  |  |
| 20 |  |  |  |  |  |  |
| 21 | А3 | ІК12.310БАК.006 Д6 |  | 1 |  |  |
| 22 |  |  |  |  |  |  |
| 23 | А3 | ІК12.310БАК.006 Д7 |  | 1 |  |  |
| 24 |  |  |  |  |  |  |
| 25 |  |  |  |  |  |  |
| 26 |  |  |  |  |  |  |
| 27 |  |  |  |  |  |  |
| 28 |  |  |  |  |  |  |

**Пояснювальна записка**

**до дипломного проєкту**

**на тему: «Підсистема автоматизованого керування складськими роботами»**

Київ — 2025

Зміст

[Календарний план3](#_Toc200292858)

[**Вступ** 15](#_Toc200292859)

[**РОЗДІЛ 1** 17](#_Toc200292860)

[**АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ У СФЕРІ АВТОМАТИЗАЦІЇ СКЛАДСЬКИХ СИСТЕМ** 17](#_Toc200292861)

[1.1 Загальний аналіз предметної області 17](#_Toc200292862)

[1.2 Класифікація типів складів та логістичних операцій 19](#_Toc200292863)

[1.3 Існуючі рішення для автоматизації логістичних процессів на складах 21](#_Toc200292864)

[1.4 Аналіз підходів до реалізації навігації 23](#_Toc200292865)

[1.5 Очікуваний ефект від впровадження автоматизації 25](#_Toc200292866)

[1.6 Висновок до розділу 1 26](#_Toc200292867)

[**РОЗДІЛ 2 27**](#_Toc200292868)

[**ПОСТАНОВКА ЗАДАЧІ ТА ВИЗНАЧЕННЯ ВИМОГ 27**](#_Toc200292869)

[2.1 Постановка задачі 27](#_Toc200292870)

[2.2 Технічні вимоги до проєктування 27](#_Toc200292871)

[2.2.1 Архітектурні принципи 27](#_Toc200292872)

[2.2.1.1 Модульність 27](#_Toc200292873)

[2.2.1.2. Масштабованість та потокобезпечність 28](#_Toc200292874)

[2.2.1.3. Відмовостійкість 29](#_Toc200292875)

[2.2.2 Алгоритмічні рішення 30](#_Toc200292876)

[2.3 Функціональні та нефункціональні вимоги до системи 31](#_Toc200292877)

[2.3.1. Функіональні вимоги системи 32](#_Toc200292878)

[2.3.2 Нефункціональні вимоги 33](#_Toc200292879)

[2.4 Виснок до розділу 2 35](#_Toc200292880)

[**РОЗДІЛ 3** 36](#_Toc200292881)

[**ПРОЄКТУВАННЯ** 36](#_Toc200292882)

[3.1 Загальна архітектура системи 36](#_Toc200292883)

[3.2 Проєктування Бази даних 38](#_Toc200292884)

[3.2.1 Вступ 38](#_Toc200292885)

[3.2.2 Опис таблиць 39](#_Toc200292886)

[3.2.3 Зв’язки між таблицями бази даних 42](#_Toc200292887)

[3.2.4 Висновок до розділу 3 43](#_Toc200292888)

[3.3 Проєктування структури складу 44](#_Toc200292889)

[3.3.1 Визначення розмірів складу та координатної сітки 45](#_Toc200292890)

[3.3.2 Зони складу (полиці, палети, зарядки) 46](#_Toc200292891)

[3.3.3 Логіка проходів 48](#_Toc200292892)

[3.3.4 Вимоги до адаптації структури 49](#_Toc200292893)

[3.3.5 Виснок по структурі складу: 50](#_Toc200292894)

[3.4 Проєктування обробки замовлень 51](#_Toc200292895)

[3.4.1 Структура замовлення та його компоненти 51](#_Toc200292896)

[3.5.2 Життєвий цикл замовлення 52](#_Toc200292897)

[3.5.3 Перевірка доступності товарів 53](#_Toc200292898)

[3.5.4 Резервування полиць 54](#_Toc200292899)

[3.5.4.1 Правила сумісності товарів на полиці 54](#_Toc200292900)

[3.5.4.2 Атомарне резервування 54](#_Toc200292901)

[3.5.4.3 Зміна статусу полиці 55](#_Toc200292902)

[3.5.5. Висновок 55](#_Toc200292903)

[3.6 Проєктування навігації та уникнення колізій 56](#_Toc200292904)

[3.6.1 Архітектура навігаційної підсистеми 56](#_Toc200292905)

[3.6.2 Алгоритми планування маршруту 57](#_Toc200292906)

[3.6.2.1 Основні вимоги до алгоритмів 57](#_Toc200292907)

[3.6.2.2 Обрані алгоритми 57](#_Toc200292908)

[3.6.3 Виявлення та уникнення колізій 60](#_Toc200292909)

[3.6.5 Адаптивна навігація та обхідні шляхи 62](#_Toc200292910)

[3.6.5.1 Пошук альтернативних позицій підходу 63](#_Toc200292911)

[3.6.5.2 Динамічне очікування звільнення позицій 63](#_Toc200292912)

[3.6.5.3. Випробування обхідних маршрутів з обмеженим часом 63](#_Toc200292913)

[3.7. Проєктування логіки зарядки 64](#_Toc200292914)

[3.7.1 Принципи роботи енергетичного моніторингу 64](#_Toc200292915)

[3.7.2 Оцінка доцільності виконання замовлення 65](#_Toc200292916)

[3.8 Висновки по розділу проєктування 66](#_Toc200292917)

[**РОЗДІЛ 4.** 68](#_Toc200292918)

[**РЕАЛІЗАЦІЯ** 68](#_Toc200292919)

[4.1 Архітектура програмної реалізації 68](#_Toc200292920)

[4.1.1 Вибір технологій 68](#_Toc200292921)

[4.1.2 3агальний опис реалізації архітектури підсистеми 68](#_Toc200292922)

[4.1.3. Принципи модульного проєктування та розподілу відповідальності: 69](#_Toc200292923)

[4.2 Реалізація цифрової сітки складу та адмін—панелі 70](#_Toc200292924)

[4.3 Механізм створення симуляційного замовлення 72](#_Toc200292925)

[4.4 Логіка взаємодії робота з замовленням 74](#_Toc200292926)

[4.4.1 Опис фіксованих умов та обмежень системи 74](#_Toc200292927)

[4.4.2 Призначення замовлення (вибір, перевірка, закріплення) 76](#_Toc200292928)

[4.4.3 Взаємодія з інвентарем (палети, товари) 77](#_Toc200292929)

[4.4.4 Взаємодія з полицями (резервування, доставка) 80](#_Toc200292930)

[4.5 Навігація та уникнення колізій 81](#_Toc200292931)

[4.5.1 Реалізовані алгоритми пошуку шляху та інтеграція у логіку руху 81](#_Toc200292932)

[4.5.2 Детекція і розв’язання deadlock—ситуацій 84](#_Toc200292933)

[4.5.2.1 Аналіз залежностей 85](#_Toc200292934)

[4.5.2.2 Опис вибору пріоритетів між роботами 86](#_Toc200292935)

[4.5.2.3 Стратегія відходу 88](#_Toc200292936)

[4.6 Система керування батареєю і зарядкою 90](#_Toc200292937)

[4.6.1. Моделювання реального споживання енергії під час руху 91](#_Toc200292938)

[4.6.2. Прийняття рішення про необхідність зарядки 93](#_Toc200292939)

[4.6.4. Процес зарядки: фізична модель та алгоритмічна реалізація 95](#_Toc200292940)

[4.6.5 Демонстрація реалізованой механіки зарядки 97](#_Toc200292941)

[4.7 Моделювання та опис результатів 100](#_Toc200292942)

[4.8 Висновок до розділу 4 103](#_Toc200292943)

[**РОЗДІЛ 5.** 105](#_Toc200292944)

[**ВИСНОВКИ** 105](#_Toc200292945)

# **Вступ**

Дипломна робота присвячена розробці та впровадженню інтелектуального алгоритму навігації для мобільних роботів у складському середовищі. На мій власний погляд автоматизація складських процессів роботами, це не лише шлях до підвищення ефективності, а й відповідь на проблеми, пов’язані з людським фактором, оптимізацією ресурсів та необхідністю швидкої адаптації до змін у сучасному попиті. Рішення зосередитися на розробці такої інтелектуальної підсистеми прийшло до мене, коли я отримав власний досвід як працівник типових складських систем, де зрозумів, що такі системи є в край не надійними, та не ефективними. Тому створення мобільної автоматизованої підсистеми яка здатна працювати у складних, динамічних та обмежених умовах, здається мені достатньо виправданою.У межах проєкту буде створено комплексне програмне забезпечення, яке моделює роботу автономних складських роботів у багатокомпонентній динамічній системі. Основна увага буде приділятися інтеграції існуючих ефективних алгоритмів пошуку шляху, а також впровадженню механізмів автоматичного уникненню колізій мобільних агентів, що на мою думку, буде одною з найважчих викликів у роботі.

Створення алгоритму уникнення колізій , гарантує забезпечення підсистеми – надійною навігацією в умовах обмеженого простору та високої щільності агентів. Особливу складність становить моделювання реалістичних сценаріїв, де роботи повинні не лише знаходити оптимальні маршрути, а й ефективно взаємодіяти між собою, уникати блокувань, своєчасно реагувати на зміни у середовищі.

Розроблений прототип буде не лише імітувати ключові аспекти автоматизації складу, а й створювати базову платформу для подальшой інтеграції у сфері багатоагентних систем. Система повинна дозволяти експериментувати з різними стратегіями планування руху, аналізувати ефективність алгоритмів у різних сценаріях навантаження та досліджувати питання масштабованості й надійності. Отриманий програмний застосунок може в подальшому інтегруватися у реальну підсистему складу. Завдяки розробленій підсистемі, алгоритми пошуку шляху будуть надійно виконувати поставлені задачі в різних умовах та залишатися ефективними і надійними. Важно зазначити, що я з самого початку буду моделювати максимально складні умови для роботів-агентів, про що свідчить вузька сідка складу з обмеженим простором. Такі умови дають гарантію, що при успішному тестуванні, використання в справжніх умовах буде не менш надійним.

# **РОЗДІЛ 1**

## **АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ІСНУЮЧИХ РІШЕНЬ У СФЕРІ АВТОМАТИЗАЦІЇ СКЛАДСЬКИХ СИСТЕМ**

### 1.1 Загальний аналіз предметної області

На сьогоднішний день управління логістичними процесами на складі є складною та багатоетапною задачею. Основні процеси, такі як: отримання, зберігання, сортування, переміщення, комплектація та відвантаження товарів створюють велике навантаження на робочий персонал знижуючи ефективність роботи. А ефективність процесів безпосередньо впливає на швидкість виконання замовлень, рівень обслуговування клієнтів та загальну продуктивність підприємства. А це в свою чергу на задоволеність клієнтів від наданого сервісу. У типових складських системах ці операції виконуються вручну (тобто людьми) зазвичай за допомогою напівавтоматизованої техніки. У такого підхода є великі мінуси, які пов'язані з високими витратами часу, людськими ресурсами і на моєму досвіді дуже часто призводить до помилок та перевантаження працівників і звісно зниження ефективності роботи складу в цілому.

Аналізуючи сучасний стан споживчого попиту, можна зробити висновок, що є активне зростання обсягів торгівлі та підвищення вимог до швидкості доставки. Тому все більше компаній звертаються до повністю автоматизованих рішень. Однією з найперспективніших технологій у цій сфері є автономні мобільні роботи (Autonomous Mobile Robots або AMR), які здатні мінімізувати людське втручання в логістичні процеси на складі, самостійно пересуватися територією складу, виконувати завдання з транспортування товарів, взаємодіяти з іншими елементами інфраструктури складу та приймати рішення в реальному часі. Згідно з даними звіту(таблиця 1.1) GlobeNewswire, ринок AMR для логістики та складування оцінювався в 2,96 мільярда доларів США у 2023 році та очікується, що до 2032 року він досягне 18,56 мільярда доларів США, що свідчить про значне зростання попиту на ці технології. [[1]](#_Джерела)

Таблиця 1.1 — Характеристика ринку

| **Характеристики ринку** | **Деталі** |
| --- | --- |
| **Розмір ринку у 2023 році** | 2,96 млрд доларів США |
| **Розмір ринку до 2032 року** | 18,56 млрд доларів США |
| **Середньорічний темп зростання (CAGR)** | 22,6% з 2024 по 2032 рік |
| **Базовий рік** | 2023 |
| **Прогнозний період** | 2024—2032 |
| **Історичні дані** | 2020—2022 |

Такі роботи зазвичай оснащені сенсорами, такі як: камери, лідари, комп'ютерний зір, модуль зв’язку. Однак для повноцінного виконання своїх функцій роботи мають бути інтегровані у складну програмну систему, яка відповідає за маршрутизацію, управління завданнями, розподіл замовлень, контроль за зарядом батареї, уникнення колізій та оптимізацію взаємодії між агентами. Програмне забезпечення виконує роль «мозку» роботизованої системи та має враховувати велику кількість факторів, зокрема пріоритети завдань, наявність ресурсів, поточну позицію кожного робота, змінні умови середовища та інші обмеження.

Одна з основних проблем у багатороботних системах є забезпечення узгодженої поведінки роботів, щоб уникнути конфліктів при одночасному доступі до тих самих ресурсів. Це стосується як фізичного простору (наприклад, двоє роботів не повинні одночасно опинятись в одній позиції, або уникненню колізій та тупикових ситуацій де алгоритми декілька роботів «зависають»), так і логічного доступу до даних (наприклад, уникнення ситуації, коли два роботи беруть одне й те саме замовлення або товар). Ще одним важливим аспектом є підтримка стабільності системи при збільшенні кількості роботів та замовлень. У таких умовах зростає потреба в розподілені задач, що дозволить роботам приймати рішення та працювати автономно, з розумінням дій інших роботів. Тому інтеграція інетелектуальної підсистеми відіграє ключову роль у досягненні високої ефективності багатороботних систем.

### 1.2 Класифікація типів складів та логістичних операцій

Склади відіграють ключову роль у ланцюгах постачання, забезпечуючи зберігання, облік, сортування та переміщення товарів. Існує кілька класифікацій складів за різними ознаками, ми розглянемо основну ознаку для нашої роботи, а саме класифікація складів за ступенем автоматизації.

1.2.1 Класифікація складів за ступенем автоматизації

У сучасних умовах інтенсивного розвитку логістичних ланцюгів класифікація складів за ступенем автоматизації стає надзвичайно актуальною. Вибір рівня автоматизації безпосередньо впливає на швидкість і точність виконання складських операцій, витрати підприємства, потребу в персоналі та здатність масштабування системи.

Початковий рівень складають ручні, або традиційні, склади, які практично не передбачають використання автоматизованих засобів. У таких системах прийом, переміщення, зберігання, облік і відвантаження товарів виконуються виключно вручну. Вантажно—розвантажувальні операції здійснюються за допомогою візків, рокл або навантажувачів, однак саме працівник виконує всю координаційну та контрольну функцію. Перевагою цього типу складу є низькі інвестиційні витрати та простота впровадження, проте він має низьку продуктивність і високий ризик помилок.

Другий рівень це напівавтоматизовані склади. У таких системах частина процесів, наприклад, транспортування товарів чи сканування кодів виконується за допомогою технічних засобів, однак ключові операції залишаються за працівником. Використовуються конвеєри, автоматичні штабелери, сканери та програмне забезпечення для реєстрації операцій. Це дає змогу зменшити вплив людського чинника, пришвидшити виконання типових завдань, підвищити точність обліку, але не усуває повністю залежність від персоналу.

На третьому рівні, це автоматизовані склади, які ґрунтуються на використанні інтегрованих WMS—систем, роботизованих платформ, автоматичних сортувальників та кран—штабелерів. Тут частка участі людини зводиться до мінімуму, а основні процеси відбуваються автоматично. В таких системах реалізуються стратегії типу «товар до людини», що дозволяє зменшити час пересування працівників та підвищити ефективність збору замовлень.

Найвищий рівень це повністю роботизовані склади, які функціонують майже без участі людини. У таких системах автономні мобільні роботи (AMR), автоматизовані стелажні комплекси (AS/RS), роботизовані маніпулятори та системи комп’ютерного зору забезпечують виконання всіх ключових завдань: зберігання, переміщення, інвентаризації, пакування та підготовки до відправки. Робота таких складів синхронізується з корпоративними системами управління (ERP, CRM), що дозволяє досягати максимальної точності, швидкості та адаптивності до змін. Автор посібника підкреслює: «Таким чином, застосування інтелектуальних систем і технологій, заснованих на імітаційному моделюванні, свідчить про те, що вона є досить актуальною, оскільки є частиною інтелектуальних інформаційних систем, забезпечуючи при цьому гнучкість системи, спрямованої на підвищення ефективності управління, організації та контролю над усіма сполучними елементами і об’єктами складської інфраструктури споживчого ринку регіону.» (с. 10).

Хоча запровадження автоматизованих рішень потребує значних фінансових ресурсів і технічної підготовки, їхні переваги — зниження витрат на персонал, збільшення продуктивності, зменшення помилок та вища гнучкість роблять їх актуальними для підприємств, які прагнуть конкурувати у швидкозмінному середовищі електронної комерції та глобальних поставок.

Таким чином, ступінь автоматизації складу є важливим індикатором технологічної зрілості логістичної системи підприємства та його готовності до цифрової трансформації.

### 1.3 Існуючі рішення для автоматизації логістичних процессів на складах

Автономні мобільні роботи (AMR) стали основою новітніх підходів до автоматизації складських і виробничих процесів. Відмінною рисою AMR є здатність самостійно пересуватися в динамічному середовищі без потреби у фіксованій інфраструктурі, такій як напрямні рейки або магнітні смуги. Завдяки використанню сенсорів, камер, лідарів, GPS—навігації та алгоритмів побудови маршруту, AMR можуть адаптуватися до змін у середовищі та уникати перешкод у реальному часі.

На практиці AMR знаходять застосування в таких рішеннях, як автоматизоване транспортування вантажів на складі, підтримка збирання замовлень, переміщення компонентів між виробничими ділянками, сортування та інвентаризація. Наприклад, система Amazon Robotics використовує велику кількість AMR для доставки стелажів до працівників, мінімізуючи потребу в пересуванні людей.

Amazon Robotics це підрозділ Amazon, який займається розробкою та впровадженням робототехнічних рішень для автоматизації логістичних процесів. Одним із ключових елементів цієї системи є автономні мобільні роботи, які виконують різноманітні завдання на складах.

Один із перших роботів, впроваджених Amazon після придбання компанії Kiva Systems у 2012 році, це cаме Kiva. Ці роботи транспортують стелажі з товарами до працівників, що дозволяє зменшити час на пошук і відбір товарів. Завдяки цьому підходу Amazon змогла збільшити щільність зберігання товарів і підвищити ефективність обробки замовлень. [[2]](#_Джерела)

У 2022 році Amazon представила нового автономного мобільного робота Proteus. На відміну від попередніх моделей, Proteus здатний безпечно пересуватися в середовищі, де працюють люди, завдяки використанню передових технологій сприйняття та навігації. Цей робот може самостійно піднімати, транспортувати та опускати контейнери в логістичних центрах Amazon, взаємодіючи з людьми без потреби в обмежених зонах. Proteus використовує світлові, звукові та фізичні сигнали для комунікації з працівниками, забезпечуючи безпечну співпрацю. [[3]](#_Джерела)

З технічного боку AMR роботи можуть бути побудовані на основі централізованої(таблиця 1.2) або децентралізованої (таблиця 1.3) архітектури. У централізованій моделі навігація, маршрутизація та управління завданнями здійснюються з єдиного сервера. У децентралізованій кожен робот приймає рішення самостійно, спираючись на локальну інформацію та обмін даними з іншими агентами. У моєї реалізації було обрано централізовану модель. Для того, щоб поясними мій вибір давайте проаналізуємо позитивні на негативні сторони обох моделей:

Таблиця 1.2 — Переваги та недоліки Централізованой системи

Централізована система:

| Переваги | Недоліки |
| --- | --- |
| Простота реалізації , це вся логіка зберігається в одному місці (сервер, база даних). | Вузьке місце — сервер може стати точкою відмови (якщо він завис — система зупиняється) |
| Єдина картина світу — всі роботи бачать однакову актуальну інформацію. | Менша гнучкість — роботи не приймають рішення самостійно, кожна дія вимагає звернення до центру. |
| Зручно контролювати легко керувати пріоритетами, моніторити статуси, вводити зміни. | Складне масштабування — при великій кількості роботів збільшується навантаження на центральний вузол. |

Таблиця 1.3 — Переваги та недоліки Децентралізованой системи

Децентралізована система:

| Переваги | Недоліки |
| --- | --- |
| Висока надійність — відмова одного робота не зупиняє всю систему. | Складна реалізація — потребує реалізації протоколів обміну даними, логіки конфліктів. |
| Масштабованість — легко додавати нових роботів без перевантаження серверної частини. | Ризик колізій — без глобального погляду роботи можуть «не поділити» ресурси. |
| Гнучкість — роботи можуть адаптуватись до локальних змін у середовищі. | Складніше відлагоджувати — помилки важче виявити, бо поведінка залежить від локального контексту. |

На мій власний погляд вибір саме централізованої архітектури є цілком обґрунтований: вона дозволила сфокусуватись на логіці алгоритмів, взаємодії роботів і точній симуляції, без надлишкової складності в синхронізації між агентами. Також можно було б розглянути гібридну систему, де центральна система відповідає за високорівневе планування: розподіл замовлень, контроль стану всіх ресурсів, глобальні правила. А кожен робот автономно вирішує локальні задачі: планує маршрут, приймає рішення про обʼїзд, уникає зіткнень, виконує підхід до палети чи полиці. На мій погляд, таке рішення є найкращим, але реалізація такого рішення нажаль не можлива без великої кількості ресурсів, які я, як студент не можу мати. Та і моєю основную метою є зосередитись на розробці та інтегруванні алгоритмів пошуку шляху.

Важливо також враховувати, що ефективність AMR сильно залежить від програмного забезпечення, яке відповідає за координацію роботів, обробку конфліктів, розподіл задач і взаємодію з іншими елементами інфраструктури. Впровадження подібних рішень вимагає не лише належної технічної бази, а й детального планування логістичних процесів, адаптації під конкретну топологію складу та відповідного навчання персоналу.

### 1.4 Аналіз підходів до реалізації навігації

Ефективна навігація є критично важливою складовою функціонування автономних мобільних роботів (AMR) у складських системах. Від правильності вибору навігаційного підходу залежить здатність системи адаптуватися до змін, уникати зіткнень, забезпечувати високу продуктивність та енергоефективність.

На початкових етапах розвитку складської автоматизації використовувалися статичні підходи до навігації, коли роботи рухалися за строго заданими маршрутами. Такі маршрути будувалися за допомогою фіксованих магнітних смуг, QR-кодів на підлозі або інфрачервоних маяків. Рух роботів був обмежений наперед визначеними точками входу і виходу, а будь-яка зміна в конфігурації складу вимагала перепрограмування або фізичної реконфігурації середовища. Ці методи показали низьку гнучкість і неефективність у динамічному середовищі з частими змінами, типових для сучасної логістики.

У зв’язку з цим усе більшого поширення набувають **динамічні алгоритми навігації**, які дозволяють AMR самостійно обирати оптимальний маршрут в умовах змінного середовища. Найпопулярнішим серед них є **алгоритм A\*** . Його основною перевагою є здатність враховувати як відстань до цілі, так і вартість пересування, що дозволяє будувати найкоротший або найефективніший шлях залежно від заданих критеріїв. A\* добре підходить для середовищ із відомими перешкодами та фіксованою картою, але може бути доповнений евристиками або перераховуватись у реальному часі при зміні умов. [[7]](#_Джерела)

В умовах багатороботних систем навігація має бути **координаційною** —тобто враховувати наявність інших роботів у полі дії. Для цього використовуються модифікації алгоритмів, що забезпечують уникнення конфліктів (наприклад, MultiAgent Path Finding або MAPF), а також системи резервування маршрутів, тайм-слотів або навіть передбачення можливих тупиків і заторів. Також розробляються алгоритми пріоритезації руху залежно від завантаженості чи типу завдання.

Ще одним важливим аспектом є **енергоефективність** навігації. Оскільки роботи працюють від батарей, алгоритми повинні враховувати залишковий заряд при плануванні маршрутів, аби уникнути ситуацій, коли робот не зможе завершити завдання або заблокує шлях. У деяких реалізаціях система автоматично перенаправляє роботів на зарядні станції, щойно рівень заряду досягає критичного значення. Крім того, можуть використовуватись **алгоритми балансування навантаження** тобто рівномірного розподілу завдань між роботами, аби уникнути перевантаження одних і простою інших.

Сучасні AMR системи також активно використовують **технології SLAM (Simultaneous Localization and Mapping)** це одночасне локалізування в просторі та побудова карти. SLAM дозволяє працювати в невідомому середовищі, автоматично оновлювати карту при виявленні нових об’єктів та будувати маршрути без зовнішніх позначок.

Таким чином, підхід до навігації роботів визначається складністю середовища, кількістю учасників (роботів), динамікою змін і вимогами до продуктивності. Перехід від статичних методів до гнучких, динамічних і адаптивних алгоритмів навігації є ключовим етапом у розвитку інтелектуальних складських систем.

### 1.5 Очікуваний ефект від впровадження автоматизації

Впровадження автоматизованих систем управління складом, з використанням автономних мобільних роботів, має значний економічний вплив на підприємства, що здійснюють логістичну діяльність. Одним із ключових ефектів є скорочення витрат на персонал, оскільки частина рутинних складських операцій, таких як: транспортування, сортування, зберігання виконується без участі людини. Це дозволяє зменшити залежність від людського фактору, знизити рівень помилок при виконанні завдань, а також зменшити витрати на навчання та управління персоналом.

Іншим важливим аспектом є підвищення продуктивності. Роботи можуть працювати цілодобово, без перерв і з мінімальними простоями, що дозволяє обробляти більше замовлень за одиницю часу. Як вказав автор посібника «В компанії Amazon роботи, які займаються транспортуванням замовлень на складах, виявилися в 4—5 разів ефективніше працівників, які працюють на складі. Ті операції, на які людина витрачає 60 — 75 хв., робот виконує за 15 хв. Роботи дозволили знизити операційні витрати кожного складу приблизно на 20 % (22 млн. дол.). За оцінками фахівців Deutsche Bank якщо Аmazon задіє роботів на всіх 110 складах, вони дозволять компанії заощадити близько 2,5 млрд. дол. Незважаючи на те, що установка роботів в кожному складі обійдеться в 15 — 20 млн. дол., одноразова еконо мія складе близько 800 млн. .» с.239 [[5]](#_Джерела)

Додатково, за рахунок точного моніторингу запасів у реальному часі, автоматизація сприяє зменшенню обсягів «мертвих» запасів і покращенню планування закупівель. У сукупності всі ці фактори забезпечують швидке повернення інвестицій. За оцінками багатьох компаній, термін окупності впровадження AMR—системи становить від 1,5 до 3 років. [[6]](#_Джерела)

Таким чином, автоматизація складських процесів є не лише технічним покращенням заради покращеня, але й ефективним економічним інструментом, що дозволяє підприємствам підвищити конкурентоспроможність на ринку та адаптуватися до викликів сучасної логістики.

### 1.6 Висновок до розділу 1

У межах дипломної роботи предметну область було детально проаналізовано з позиції розробки програмної моделі поведінки та координації складських роботів. Проведений аналіз дозволив чітко визначити функціональні та не функціональні вимоги до системи в наступному етапі, зрозуміти як сформувати структуру бази даних, окреслити типові сценарії взаємодії та описати технічні обмеження, які потрібно враховувати під час розробки. Це заклало основу для створення симуляційного середовища, в якому можна буде тестувати різні алгоритми, порівнювати їх ефективність та готувати систему до можливої інтеграції з реальними фізичними агентами.

# **РОЗДІЛ 2**

# **ПОСТАНОВКА ЗАДАЧІ ТА ВИЗНАЧЕННЯ ВИМОГ**

### 2.1 Постановка задачі

Розроблена програмна підсистема повинна моделювати роботу складу продовльчого онлайн магазин з автономними мобільними роботами, що виконують обробку замовлень без участі людини. Включати механізми генерації замовлень, автоматизовану логіку їх виконання, побудову маршрутів руху роботів, облік інвентарю, а також графічну візуалізацію поточних процесів на складі. Архітектура повинна бути побудована таким чином, щоб забезпечити модульність, масштабованість та можливість реального впровадження в майбутньому.

Ключовим компонентом проєкту повинна бути логіка координації дій між роботами, зокрема: навігація з уникненням колізій, контроль заряду батареї, синхронізація доступу до замовлень і поличок, резервування клітинок сітки для запобігання конфліктам маршруту. Програмна система пов’язана з реляційною базою даних, яка зберігає всю динамічну інформацію: положення роботів, статуси замовлень, вміст інвентарю, доступність полиць тощо.

## 2.2 Технічні вимоги до проєктування

### 2.2.1 Архітектурні принципи

#### 2.2.1.1 Модульність

Система повинна бути побудована за принципом модульної архітектури, де кожен компонент має чітко визначену відповідальність і мінімальні залежності від інших.

* Модуль логіки роботів: інкапсулює всю поведінку робота — від базової навігації до координації з іншими агентами.
* Модуль навігації: є повністю автономним — його можна замінити або вдосконалити без впливу на інші частини системи.
* Модуль обробки замовлень: управляє чергою задач та контролює життєвий цикл обробки кожного замовлення.
* Модуль симуляції: відповідає за візуалізацію та тестування роботи всієї системи.

Модульна архітектура надає системі низку важливих переваг, зокрема можливість незалежної розробки, тестування та налагодження кожного окремого компонента без впливу на роботу всієї системи. Завдяки чіткому розмежуванню функцій, внесення змін до одного модуля не порушує стабільність інших частин програми, що значно полегшує супровід і розвиток. Крім того, така структура сприяє простому масштабуванню: нові функції можуть додаватися без необхідності модифікувати існуючі модулі. Ще однією перевагою є можливість повторного використання розроблених компонентів в інших проєктах або в нових контекстах, що підвищує ефективність розробки загалом.

#### 2.2.1.2. Масштабованість та потокобезпечність

Архітектура системи повина бути спроєктована з урахуванням необхідності підтримки паралельної роботи великої кількості автономних роботів, що діють незалежно один від одного. Кожен робот функціонує у власному потоці виконання, що дозволяє системі динамічно масштабуватись додавання або видалення роботів не потребує перезапуску або змін у логіці інших агентів. Це забезпить гнучкість і адаптивність системи до змін у навантаженні, зокрема при зростанні кількості замовлень чи роботів.

Особливу увагу треба приділити потокобезпечності при роботі зі спільними ресурсами — такими як глобальна карта складу, інформація про зарезервовані клітинки, черги завдань чи статуси об'єктів. Для синхронізації доступу до цих ресурсів використовуються механізми блокування (наприклад, Lock), що дозволяє уникнути конфліктів під час одночасного звернення кількох потоків.

Крім того, глобальні структури даних у системі потребують оптимізованості для швидкого доступу та ефективного оновлення. Зокрема, для резервування маршрутів і моніторингу цілей використовуються хеш—таблиці, які забезпечують постійний час доступу незалежно від загальної кількості роботів. Такий підхід гарантує збереження високої продуктивності системи навіть у випадках масштабного розгортання з великою кількістю агентів.

Завдяки поєднанню масштабованої багатопотокової архітектури та надійної синхронізації, система залишається стабільною, гнучкою та готовою до розширення під нові вимоги або сценарії використання.

#### 2.2.1.3. Відмовостійкість

Система повинна мати механізми, що забезпечують її стійкість до типових збоїв, які можуть виникати під час автономної роботи мобільних роботів у динамічному складському середовищі. Надійність системи є критично важливою умовою для забезпечення безперервності логістичних процесів і мінімізації ризиків зупинки або втрати ефективності.

Передусім, система повинна мати функціонал для **виявлення навігаційних помилок**, зокрема таких, як недосяжність цільової позиції внаслідок змін у середовищі або блокування маршруту іншими агентами. У разі неможливості побудувати маршрут, система має автоматично ініціювати спробу повторного пошуку з використанням альтернативного алгоритму або повідомити модуль управління для подальших дій.

Також необхідно передбачити обробку ситуацій, коли **товар виявляється відсутнім** на палеті або замовлення стає **неактуальним** через його скасування або зміни. У таких випадках система повинна мати логіку перевірки стану товару перед ініціацією дій і забезпечувати гнучке перепризначення задач іншим роботам або перенаправлення на альтернативні позиції.

Крім того, важливо, щоб система могла **виявляти конфліктні ситуації між роботами**, зокрема спробу двох або більше агентів зайняти одну й ту саму клітинку. Для цього необхідно реалізувати аналіз можливих deadlock—ланцюгів, а також стратегії розв’язання конфліктів: відступ, надання пріоритету окремим агентам, затримка дій або побудова обхідного маршруту.

Таким чином, система повинна мати продуману логіку виявлення збоїв і реагування на них у реальному часі, що є ключовою умовою забезпечення її стабільності та функціонування в умовах невизначеності та високої навантаженості.

### 2.2.2 Алгоритмічні рішення

Алгоритм пересування автономних роботів на складі має забезпечувати ефективне, безпечне та скоординоване переміщення кількох агентів у спільному середовищі. Основна мета, це дозволити кожному роботу знаходити шлях до цілі (палети або полиці), переміщатися крок за кроком, уникати конфліктів із іншими роботами, своєчасно заряджатися і виконувати завдання, не створюючи заторів.

Передусім необхідно реалізувати механізм пошуку шляху. Робот повинен мати змогу розрахувати маршрут з урахуванням поточних перешкод та зайнятих клітинок, меж сітки, палет, полиць і маршрутів інших роботів. Варто реалізувати два базові алгоритми, та систему збору статистики для кожного з них, щоб згодом мати змогу обирати ефективніший варіант в конкретних умовах.

Під час переміщення робот має перевіряти, чи не зайнята наступна клітинка. Якщо клітинка зайнята іншим роботом або вона є ціллю іншого агента, рух призупиняється. Для цього необхідно впровадити глобальну систему резервування клітинок, яка унеможливлює одночасний доступ до однієї й тієї самої координати.

Окрему увагу потрібно приділити виявленню та вирішенню deadlock—ситуацій. Якщо робот очікує, поки звільниться клітинка, і ця клітинка, у свою чергу, є ціллю іншого робота, який теж заблокований, формується ланцюг залежностей. Алгоритм має вміти будувати таку ланцюгову послідовність та визначати, чи є вона циклічною. Якщо виявлено deadlock, робот із вищим пріоритетом (наприклад, із більшим ID) має спробувати відійти знайти найближчу вільну клітинку, тимчасово відступити й дати іншим дорогу. Якщо жоден робот не може відступити, система має спробувати перерахувати маршрут або зробити паузу, після чого повторити спробу.

У випадку тривалого блокування робот має спробувати побудувати альтернативний маршрут, уникаючи не лише зайнятих клітинок, а й тих, що знаходяться поруч із ними через додаткові штрафи до ваги маршруту.

Також потрібно реалізувати моніторинг заряду батареї. Якщо рівень опускається нижче порогового значення, робот має перервати виконання поточного завдання і самостійно вирушити до зарядної станції. Процес заряджання повинен бути асинхронним і поступовим, після чого робот повертається до роботи.

Крім логіки руху, система має працювати з базою даних: отримувати координати роботів, статуси, рівень заряду, дані про замовлення, палети, полиці та оновлювати їх після кожного кроку. Варто також реалізувати механізм діагностики поточного стану робота — положення, заряд, статус, маршрут, тощо для візуалізації або логування.

Таким чином, майбутній алгоритм має вирішувати одразу кілька задач: пошук маршруту, уникнення конфліктів, обхід перешкод, вирішення deadlock, динамічне перепланування, контроль заряду, взаємодію з БД і підтримку безперервної роботи кількох роботів у спільному просторі без збоїв.

## 2.3 Функціональні та нефункціональні вимоги до системи

Підсумовуючи, вважаю доречним визначити функіональні функціональні та нефункціональні вимоги. Вимоги до програмної системи поділено на функціональні та нефункціональні. Функціональні вимоги визначають основні можливості, які має реалізовувати система, а нефункціональні обмеження та критерії якості до системи, почнемо з функціольних вимог c. 67 [[8]](#_Джерела).

### 2.3.1. Функіональні вимоги системи

а) Управління роботами

* Система повинна підтримувати одночасну роботу до 10 роботів;
* Кожен робот повинен мати унікальний ідентифікатор та закріплену зарядну станцію;
* Роботи повинні автономно переміщуватися по складу згідно з розрахованими маршрутами;
* Система повинна відстежувати поточне місцезнаходження кожного робота в режимі реального часу.

б) Обробка замовлень

* Система повинна автоматично генерувати тестові замовлення для симуляції;
* Замовлення повинні автоматично розподілятися між доступними роботами;
* Система повинна підтримувати паралельну обробку декількох замовлень;
* Роботи повинні автоматично забирати товари з палет та доставляти їх на полиці;
* Система повинна відстежувати статуси замовлень (pending, processing, done, failed, partial).

в) Навігація та планування маршрутів

* Повинна бути можливість автоматичного вибору оптимального алгоритму;
* Система повинна запобігати колізіям між роботами через резервування клітинок;
* Повинна бути реалізована логіка виявлення та розв'язання deadlock—ситуацій;
* Роботи повинні знаходити альтернативні маршрути при блокуванні основного шляху.

г) Управління енергоспоживанням

* Система повинна відстежувати рівень заряду батареї кожного робота;
* Роботи повинні автоматично йти на зарядку при критично низькому заряді або при простої;
* Роботи повинні оцінювати достатність заряду для виконання замовлення перед його прийняттям;
* Система повинна підтримувати переривання зарядки при появі високопріоритетних замовлень.

ґ) Управління інвентарем

* Система повинна відстежувати залишки товарів на палетах та полицях;
* Повинна автоматично оновлюватися інформація про переміщення товарів;
* Система повинна резервувати полиці для конкретних замовлень;
* Повинна забезпечуватися сумісність товарів на одній полиці (тільки з одного замовлення).

д) Збір статистики та звітність

* Система повинна збирати статистику роботи кожного алгоритму навігації;
* Повинна вестися статистика успішності виконання замовлень;
* Система повинна генерувати детальні звіти з результатами симуляції.

е) Інтеграція з базою даних

* Система повинна зберігати всі дані про стан складу в реляційній БД;
* Повинна забезпечуватися атомарність операцій з БД;
* Система повинна підтримувати одночасний доступ декількох роботів до БД.

### 2.3.2 Нефункціональні вимоги

а) Продуктивність

* Час розрахунку маршруту не повинен перевищувати 100 мс для сітки 20×41;
* Система повинна підтримувати одночасну роботу до 10 роботів без деградації продуктивності;
* Час відгуку на запити до БД не повинен перевищувати 50 мс;
* Deadlock—ситуації повинні розв'язуватися протягом 10 секунд.

б) Надійність

* Система повинна забезпечувати коректну роботу при відмові одного з роботів;
* Повинна бути можливість відновлення роботи після збоїв без втрати даних;
* Система повинна автоматично обробляти помилки навігації та повторювати спроби.

в) Масштабованість

* Архітектура повинна дозволяти додавання—видалення нових роботів без зміни основного коду;
* Система повинна підтримувати розширення функціональності через додавання нових модулів;
* Повинна бути можливість зміни розмірів складу без кардинальних змін

г) Підтримуваність

* Код повинен бути структурований за модульним принципом;
* Повинна бути детальна документація основних компонентів;
* Система повинна мати зрозумілу систему логування та діагностики.

ґ) Сумісність

* Система повинна працювати на Windows з Python 3.7+;
* Повинна підтримуватися робота з MS SQL Server;

## 2.4 Виснок до розділу 2

Таким чином, визначені функціональні та нефункціональні вимоги окреслюють чітке технічне бачення проєктованої системи. Функціональні вимоги охоплюють ключові аспекти її роботи, а саме від управління роботами та обробки замовлень до навігації, енергозбереження й інтеграції з базою даних. Натомість нефункціональні вимоги встановлюють стандарти якості: продуктивність, надійність, масштабованість, підтримуваність та зручність використання. Сукупно ці вимоги створюють надійне підґрунтя для подальшого проєктування архітектури системи, її реалізації та тестування, а також дозволяють забезпечити очікувану ефективність та стабільність роботи в умовах складного динамічного середовища.

# **РОЗДІЛ 3**

# **ПРОЄКТУВАННЯ**

Метою проєктування є створення програмної підсистеми для симуляції роботи складу продовольчого онлайн магазину з автономними мобільними роботами, яка дозволяє моделювати повністю автоматизоване виконання замовлень без участі людини. Система має забезпечувати інтелектуальну координацію роботів у рамках багатоагентної взаємодії, включаючи виявлення та вирішення конфліктів і deadlock—ситуацій. Передбачено використання алгоритмів пошуку шляху з можливістю автоматичного вибору найефективнішого варіанту для кожної ситуації. Також важливим завданням є реалізація механізмів управління ресурсами зокрема, контроль рівня заряду батарей, резервування полиць та клітинок руху для уникнення колізій. Окрім цього, система повинна збирати детальну статистику роботи, що дає змогу оцінювати продуктивність і виявляти шляхи для подальшої оптимізації.

## 3.1 Загальна архітектура системи

Система управління автономними роботами проєктується як модульна та масштабована, з чітко визначеними функціями кожного компонента. Основна ідея полягає у поділі системи на логічно незалежні модулі, які відповідають за різні аспекти роботи: управління роботами, навігація, обробка замовлень, моделювання структури складу, діагностика та взаємодія з базою даних. Така архітектура сприяє легкому супроводу та подальшому розширенню системи, дозволяє уникати дублювання функцій і помилок у логіці.(кресленник ІК12.310БАК.006 Д1)

Модуль управління роботами відповідає за реалізацію автономної поведінки кожного агента. Він ініціалізує параметри, координує роботу з іншими модулями, приймає замовлення на обробку, обирає маршрут, виявляє та вирішує конфлікти руху, а також відстежує стан батареї. Для ефективної координації між роботами передбачено використання глобальних структур даних, які дозволяють резервувати клітинки на карті складу і мінімізувати ризик колізій.

Навігаційний модуль є самостійним компонентом, що відповідає за побудову маршрутів. Він реалізує різні алгоритми пошуку шляху, та автоматично перемикається між ними на основі показників ефективності. Додатково модуль підтримує механізми зворотного виклику для перевірки зайнятості клітинок, збирання статистики та оцінювання продуктивності.

Модуль обробки замовлень керує створенням, та завершенням замовлень. Він забезпечує атомарність дій при паралельному доступі кількох роботів, управляє чергою замовлень, стежить за їхнім статусом і резервує полиці під конкретне замовлення. Завершення кожного замовлення супроводжується оновленням записів у базі даних.

Для контролю функціонування системи впроваджується діагностичний модуль. Його завдання полягає у зборі даних про діяльність роботів, хід виконання алгоритмів, рівень заряду батареї, а також у формуванні звітів за результатами симуляцій. Такі звіти зберігаються у відовідному текстовому файлі проекту.

Окремий компонент відповідає за конфігурацію складу. Цей модуль задає параметри сітки, координати палет, полиць, зон заряджання. Всі ці дані інтегруються в загальну карту, яка стає доступною для інших модулів. Структура карти використовується для навігації, перевірки коректності переміщення роботів, а також для ініціалізації середовища при запуску симуляції.

Інтерфейс доступу до бази даних реалізується як окремий модуль. Він відповідає за ініціалізацію з'єднання, обробку транзакцій, забезпечення потокобезпечного доступу до таблиць та автоматичне повторне підключення у разі збоїв. Завдяки цьому забезпечується цілісність даних навіть при високому навантаженні системи.

У системі передбачено окремі елементи для запуску та тестування. Створюються стартові скрипти, які відповідають за завантаження конфігурацій, ініціалізацію всіх модулів, запуск потоків роботів, генерацію замовлень та збір статистики. Таким чином, розробники мають змогу оцінювати ефективність роботи системи під час тестування у різних сценаріях.

Взаємодія між модулями здійснюється через чітко визначені інтерфейси. Центральним сховищем усіх даних виступає база даних, яка містить інформацію про інвентар, історію замовлень, стан роботів.

Проєктування системи орієнтоване на масштабування: передбачається можливість легкого додавання нових роботів, розширення карти складу. Такий підхід дозволяє адаптувати систему до потреб реального виробництва і забезпечити її довготривалу ефективність.

## 3.2 Проєктування Бази даних

### 3.2.1 Вступ

Дуже важливим викликом у проєктуванні кожної системи є проєктування бази даних, яка забезпечує зберігання, цілісність, доступність та актуальність усіх критичних даних, пов’язаних з роботою системи. Особливо це стосується складських підсистем, де в реальному часі необхідно відслідковувати стан запасів, переміщення товарів, виконання замовлень, розташування роботів, енергоспоживання та інші параметри. У випадку автономної роботизованої системи керування складом, база даних відіграє роль центрального інформаційного вузла, що забезпечує синхронізацію між усіма модулями.

Як було сформульовано у постановці задачі, система повинна обробляти десятки замовлень одночасно, координувати дії до 15 роботів, управляти динамічними змінами в карті складу, а також зберігати статистику ефективності навігаційних алгоритмів. Усі ці дані повинні бути структуровані, узгоджені та захищені від втрати при збої. Саме тому базу даних проєктують як реляційну, що дозволяє ефективно виконувати складні запити, забезпечує зв’язки між сутностями (наприклад, замовлення — товари —полиці — роботи) та гарантує підтримку транзакційності.

База даних має забезпечити централізоване управління інформацією про:

* Поточні замовлення та їхній статус;
* Оновлювати статуси полиць;
* Складські запаси на палетах та полицях;
* Позиції кожного робота;
* Стан заряду батареї.

Крім того, база даних має забезпечити одночасний доступ декількох агентів, підтримку багатопоточних оновлень.

Таким чином, проєктування структури бази даних, це фундаментальний етап, який забезпечує надійну роботу всієї системи та дає змогу реалізувати складну логіку взаємодії між підсистемами. Тому пропоную перейти до наступного етапу, де буде розглянуто логіку побудови таблиць, визначення ключів, зв’язків та загальну ER—діаграму запропонованої моделі даних.

### 3.2.2 Опис таблиць

Структура БД яку я зараз продемонструю (таблиці 3.1 – 3.8), дозволяє ефективно моделювати та контролювати всі етапи логістичного процесу на складі: від прибуття товару до виконання замовлення. Всі таблиці логічно пов'язані, що забезпечує повну інтеграцію між підсистемами: розміщення, навігації, інвентаризації та керування завданнями для роботів.

1) Таблиця 3.1 – items (товари)

Ця таблиця містить довідник усіх товарів, які можуть зберігатися на складі.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ідентифікатор товару |
| name | varchar | Назва товару (наприклад, Coca—Cola, RedBull тощо) |
| description | varchar | Опис товару (тип напою, смак тощо) |
| packaging\_size | varchar | Кількість і об'єм в упаковці (наприклад, 12x1) |
| packaging\_type | varchar | Тип упаковки (can, glass, pet, carton, plastic тощо) |

2) Таблиця 3.2 – orders (замовлення)

Ця таблиця містить загальну інформацію про всі замовлення.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ідентифікатор замовлення |
| created\_at | datetime | Дата та час створення замовлення |
| status | varchar | Статус: pending, processing, done |

3) Таблиця 3.3 – order\_items (склад замовлень)

Пов'язує замовлення з конкретними товарами.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ідентифікатор запису |
| order\_id | int | ID замовлення |
| item\_id | int | ID товару |
| quantity | int | Кількість одиниць товару в замовленні |

4) Таблиця 3.4 – inventory (інвентаризація)

Зберігає дані про фізичне розташування товарів: чи знаходяться вони на палетах або на полицях.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ідентифікатор запису |
| item\_id | int | ID товару |
| location\_type | varchar | Тип місця зберігання: pallet або shelf |
| location\_id | int | ID палети або полиці |
| quantity | int | Кількість товару |
| x | int | Координата X на складі |
| y | int | Координата Y на складі |
| order\_id | int | ID замовлення (тільки для 'shelf', NULL для 'pallet') |

5) Таблиця 3.5 – robots (роботи)

Зберігає інформацію про кожного робота, його положення, статус і заряд батареї.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ID робота |
| name | varchar | Ім'я робота (наприклад, Robot—1) |
| status | varchar | idle, moving, charging, processing |
| x | int | Поточна координата X |
| y | int | Поточна координата Y |
| battery | int | Поточний заряд батареї (0–100%) |
| updated\_at | datetime | Час останнього оновлення |
| order\_id | int | Номер замовлення яке перевозе робот |

6) Таблиця 3.6 – pallets (палети)

Містить дані про всі палети на складі.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ID палети |
| label | varchar | Назва або позначення палети |
| x | int | Координата X |
| y | int | Координата Y |

7) Таблиця 3.7 – shelves (полиці)

Зберігає дані про полки на складі та їхнє завантаження.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ID полиці |
| shelf\_code | varchar | Умовна назва полиці (наприклад, 1—1, 3—2) |
| capacity | int | Максимальна кількість одиниць |
| status | varchar | free або busy |
| x | int | Координата X |
| y | int | Координата Y |
| order\_id | int | Якщо полиця зайнята — до якого замовлення |

8) Таблиця 3.8 – warehouse\_config(Розмір складу)

Зберігає дані сітки складу.

| Назва поля | Тип даних | Опис |
| --- | --- | --- |
| id | int | Унікальний ID |
| width | int | Ширина сладу в кількості клітинок |
| height | int | Довжина в кількості клітинок |
| Опис | varchar | Опис(наприклад: Основний склад) |

### **3.2.3 Зв’язки між таблицями бази даних**

База даних підсистеми управління складськими роботами побудована за принципами реляційної моделі та містить як фізичні (реалізовані зовнішніми ключами), так і логічні зв’язки між таблицями. Нижче наведено опис та Er—діаграму(ІК12.310БАК.006 Д2) основних взаємозв’язків між сутностями:

* 1. orders ↔ order\_items: один запис у таблиці orders може мати декілька рядків у таблиці order\_items (зв’язок «один до багатьох» через поле order\_id в order\_items).
  2. items ↔ order\_items: кожен товар із таблиці items може входити до різних рядків у order\_items (зв’язок «один до багатьох» через поле item\_id в order\_items).
  3. items ↔ inventory: таблиця inventory зберігає інформацію про кількість одиниць кожного товару та їх розміщення, зв’язуючись із таблицею items через поле item\_id.
  4. orders ↔ inventory: товари, призначені для виконання конкретного замовлення, пов’язані через поле order\_id у таблиці inventory (дає змогу ідентифікувати запаси, відкладені під замовлення).
  5. robots ↔ shelves: кожен робот (robots) обслуговує конкретну полицю (shelves), що задається полем robot\_id у таблиці shelves.
  6. orders ↔ shelves: поле current\_order\_id (або просто order\_id) у таблиці shelves дозволяє визначити, яке замовлення зараз обробляється на цій полиці.
  7. robots ↔ orders: у таблиці robots поле order\_id фіксує активне завдання (замовлення) для цього робота.
  8. shelves ↔ inventory: умовний (логічний) зв’язок полягає в тому, що якщо inventory.location\_type = «shelf» і location\_id = id полиці (shelves.id), то ця позиція інвентарю знаходиться на відповідній полиці.
  9. pallets ↔ inventory: аналогічно, коли inventory.location\_type = «pallet» і location\_id = id палети (pallets.id), то товар розміщено на відповідній палеті (цей зв’язок теж реалізується умовно, без фізичного зовнішнього ключа).

### 3.2.4 Висновок до розділу 3

У процесі проєктування бази даних основна увага приділялась тому, щоб система була зручною, логічною та придатною для реального використання на складі з автономними роботами. Структура бази включає всі необхідні таблиці, які пов’язані між собою таким чином, щоб можна було зберігати інформацію про замовлення, товари, розміщення на полицях і палетах, роботу самих роботів, а також конфігурацію складу загалом. Було важливо забезпечити не лише зберігання даних, а й їх узгодженість, саме тому використано зовнішні ключі, які допомагають підтримувати правильні зв’язки між таблицями. Окремо було передбачено гнучкість — наприклад, у таблиці інвентаря можна фіксувати, де саме зберігається товар: на полиці чи палеті, і це реалізовано через універсальний механізм із типом локації. Уся ця структура допомагає системі працювати як єдине ціле, від генерації замовлення до розміщення товару роботом у потрібному місці. У підсумку вийшла досить зрозуміла й логічна модель, яку зручно масштабувати й розвивати в майбутньому, додаючи нові функції або змінюючи логіку роботи окремих модулів.

## 3.3 Проєктування структури складу

Після проєктування структури бази даних важаю доцільним перейти до проєктування структури складу. Структури складу є одним із ключових етапів розробки підсистеми автоматизованого управління. На цьому етапі визначається логічна модель складського простору, що буде використовуватись у системі як основа для навігації, розміщення товарів, виконання замовлень та координації дій автономних агентів.

Основна мета, це створити таку топологію, яка дозволить ефективно розміщувати функціональні зони (палети, полиці, зарядні станції) та забезпечити достатній простір для переміщення роботів без перешкод. У межах цього підрозділу передбачається визначення розмірів сітки складу, поділ на зони, встановлення логічних координат для кожного елемента, а також опис призначення окремих ділянок.

Проєктування виконується з урахуванням потреб навігаційної підсистеми, яка потребує чітко заданого простору для розрахунку маршрутів, і з орієнтацією на гнучке масштабування та можливість симуляції роботи з великою кількістю роботів. Структура повинна бути логічно зрозумілою, симетричною або адаптивною, а також придатною для відображення у візуальному інтерфейсі системи.

У подальших пунктах буде розглянуто запропоновану координатну модель складу, визначено основні зони та їх взаємне розташування, а також сформульовано принципи, якими керувались при проєктуванні конфігурації складського простору.

### 3.3.1 Визначення розмірів складу та координатної сітки

Першим кроком у проєктуванні структури складу є визначення його геометричних параметрів та формування координатної системи, що буде слугувати основою для всіх подальших дій, від навігації роботів до розміщення об'єктів інфраструктури. Для моделювання буде обрано прямокутну сітку, кожна клітинка якої відповідає одній умовній позиції на складі. Такий підхід дозволяє зручно розраховувати маршрути, перевіряти зайнятість клітинок, фіксувати координати розташування палет, полиць і роботів, а також проводити облік товарів.

У даній моделі склад умовно поділено на y рядків по вертикалі (координата y) та x стовпчиків по горизонталі (координата x). Така конфігурація забезпечує достатній простір для розміщення ключових зон зберігання та організації прохідних коридорів між ними. Координати кожної клітинки подаються у вигляді пари (x, y) і є унікальними в межах складу. Це дозволяє однозначно визначити місце розташування будь—якого об’єкта.

Оскільки структура складу в цій підсистемі є симуляцією, яка за потреби може бути масштабована, сітка має умовний характер. Пропонується прийняти, що одна квадратна клітинка відповідає площі 1 квадратного метра. У разі інтеграції з реальною складською системою модель сітки буде формуватись повторно, відповідно до реальних фізичних параметрів складу та логістичних вимог. Таким чином, розміри клітинок, їхня кількість та розташування об’єктів у моделі є змінними й залежать від конкретного сценарію використання.

Запропонований підхід до формування сітки створює універсальну основу для подальшого структурування складського простору, поділу на функціональні зони та реалізації ефективної маршрутизації в умовах симульованого середовища або майбутньої адаптації до реального об’єкта.

### 3.3.2 Зони складу (полиці, палети, зарядки)

Для забезпечення структурованості складського простору та ефективної взаємодії роботів із фізичними об’єктами необхідно розподілити координатну сітку на функціональні зони. У проєктованій моделі виокремлюються три основні зони: зона палет (вхідна логістика), зона полиць (зберігання товарів після обробки замовлення) та зона зарядних станцій (відновлення енергоресурсів роботів). Розмежування цих зон дозволяє оптимізувати логістику переміщення, скоротити довжину маршрутів і зменшити кількість потенційних конфліктів під час роботи багатьох роботів одночасно.

Зона палет — це область, де товари фізично «надходять» до системи. Кожна палета(Рисунок 3.1) повинна займати окрему клітинку сітки, має унікальне розташування та ідентифікатор. Роботи здійснюють забір товарів саме з палет. Важливо враховувати зручність підходу до кожної палети з кількох напрямків, тому між палетами повинні залишатись вільні проходи.

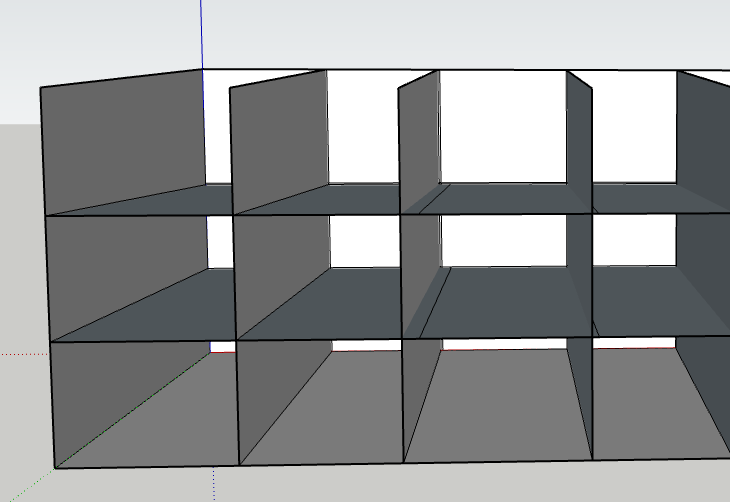


Рисунок 3.1 — стандартна палета

Зона полиць виконує роль кінцевого пункту розміщення товарів відповідно до замовлення. Полиці повинні мати координати, унікальні ідентифікатори та визначені властивості, зокрема місткість та статус (вільна, зайнята, зарезервована). У структурі бази даних кожна полиця повинна бути закріплена за конкретним замовленням, що дозволяє уникати змішування товарів і забезпечує чистоту логістичного ланцюга.

У моделі передбачається, що полиці є трирівневими — тобто містять три яруси по вертикалі, як показано на рисунку 3.2. Кожна позиція на полиці кодується відповідно до свого положення: перше число означає стовпчик, друге — ряд(ярус). Така система кодування дозволяє точно ідентифікувати місце для зберігання, що є зручним як для симуляції, так і для потенційного реального складу.

Важливо, щоб полиці фізично розміщуються у компактних рядах, між якими залишено вузькі проходи для маневрування роботів. Саме така багаторівнева конфігурація дозволяє оптимально використовувати вертикальний простір у складі, а система координат — однозначно ідентифікувати потрібну секцію для розміщення або пошуку товару. Це також дає змогу роботам точно знати, на яку висоту необхідно здійснити підйом для взаємодії з полицею, якщо у подальшій моделі буде враховано вертикальне позиціонування.



4—3

3—2

3—1

2—1

2—3

2—2

1—1

1—2

1—3

3—3

4—2

4—1

Рисунок 3.2 — Структури стелажу з полицями та їх нумерації

Зона зарядки призначена для тимчасового перебування роботів у стані підзарядки. Кожен робот повинен мати прив'язану до нього зарядну станцію, яка знаходиться в межах доступної частини сітки. Розміщення цієї зони бажано проектувати на краях або у вільній частині складу, щоб не блокувати основні логістичні потоки.

Усі ці зони мають фіксоване розташування в координатній моделі, що дозволяє використовувати їх у логіці планування маршрутів, обробки замовлень і моніторингу поточного стану системи. Чітке зонування також створює передумови для візуального відображення на симуляційному інтерфейсі та спрощує налагодження роботи багатопотокових агентів у середовищі складу.

### 3.3.3 Логіка проходів

Одним із ключових аспектів проєктування структури складу є продумана логіка проходів між об’єктами, що забезпечує безперешкодне пересування роботів у процесі виконання замовлень. В умовах автоматизованої системи, де декілька агентів працюють паралельно, ефективна організація проходів дозволяє уникнути колізій, зупинок і затримок, а також зменшити довжину маршрутів і витрати енергії.

У проєктованій симуляційній моделі було прийнято рішення розміщувати палети та полиці з урахуванням необхідних зазорів для маневрування. Наприклад, між палетами та полицями завжди передбачається хоча б один вільний ряд клітинок, який виконує роль проходу. У випадку палет, цей прохід навмисно було обмежено до лише однієї клітинки, щоб ускладнити симуляційний простір та забезпечити більш суворе тестування поведінки роботів у стиснених умовах. Таке рішення дозволяє моделювати більш реалістичні ситуації вузьких проходів, які часто трапляються на реальних складах.

Завдяки такому підходу роботи можуть підходити до об’єкта з кількох сторін, змінювати маршрут у разі блокування шляху або обирати найефективнішу траєкторію в залежності від поточної ситуації.

Проходи організовані таким чином, щоб жодна функціональна зона (палети, полиці, зарядки) не блокувала іншу, а також щоб залишалась можливість повороту, розвороту або об’їзду. Це особливо важливо у випадках, коли робот стикається з іншим агентом або знаходиться в стані низької зарядки та прямує до зарядки. Структура проходів враховує імовірні сценарії роз'їзду, резервування клітинок, та дозволяє алгоритмам пошуку шляху працювати стабільно.

Крім того, симуляційна сітка повинна підтримувати змінність конфігурації: у разі зміни вимог або масштабування, наприклад, додавання нових полиць система зберігає логіку проходів завдяки динамічному визначенню «заборонених зон», наприклад повина бути реалізована жорстка умова, що робот не може заїхати на паллету або полицю і автоматичному перерахунку доступних клітинок. У фінальній реалізації це дозволяє легко адаптувати структуру до фізичного складу з урахуванням його геометрії та логістичних потоків.

### 3.3.4 Вимоги до адаптації структури

При проєктуванні структури складу важливо передбачити можливість її гнучкої адаптації до різних сценаріїв використання — як у межах симуляції, так і для потенційної інтеграції з реальним фізичним середовищем. Оскільки розроблена система є прототипом автоматизованої логістичної підсистеми, її просторове представлення має бути масштабованим, модифікованим та відносно незалежним від конкретної конфігурації.

Передусім структура складу повинна легко змінюватись за кількістю рядів, колонок і типів зон. Наприклад, якщо буде потреба додати більше палет або полиць, система має дозволяти це без повної перебудови, а лише за рахунок оновлення конфігурації сітки та внесення нових об'єктів до бази даних. Також важливо, щоб логіка навігації роботів і маршрутизації не залежала від конкретного розташування полиць і проходів, а працювала на основі координатної моделі. Це дозволяє використовувати єдину навігаційну систему для будь—якої структури, де є мапа складу з позначеними зонами.

Ще один аспект адаптації, це **прив’язка до реальних одиниць виміру**. В моделі одна клітинка сітки умовно буде дорівнювати одному квадратному метру, але в майбутньому це співвідношення може змінюватись. Важливо, щоб система підтримувала масштабування та переналаштування без необхідності зміни алгоритмів або структури бази даних.

Підсумовуючи, забезпечення цих вимог підвищує гнучкість системи, дозволяє її використовувати як для навчальних, так і для практичних цілей у реальних складських середовищах.

### 3.3.5 Виснок по структурі складу:

На етапі проєктування сітки складу для подальшої розробки симуляційного середовища, пропонується сформувати умовну структуру складу, що охоплює всі ключові функціональні зони, необхідні для тестування логіки роботи автономних агентів. На підставі проведеного аналізу та технічних вимог до гнучкості й масштабованості моделі, склад поділено на такі основні компоненти: зону палет із продукцією, стелаж для зберігання замовлень, зону водіїв (як умовний вихід для замовлень) та окрему зарядну станцію для роботів.

Сітка реалізується у вигляді координатного поля, кожна клітинка якого умовно відповідає одному квадратному метру. Це дозволяє масштабувати модель до реальних фізичних параметрів складу. Зона палет розташована по центру, складається з окремих квадратів, які відображають індивідуальні палети з товарами, до яких роботи мають вільний доступ із кількох боків. Зліва розміщено вертикальний стелаж замовлень, що позначає полиці для остаточного складання комплектів. Справа буде розташована зона зарядки, яка є критично важливою для автономної роботи роботів із обмеженим зарядом батареї.

Запропонована структура(Рисунок 3.3) є умовною, адаптованою до задач симуляції, але з можливістю перенесення на фізичну конфігурацію складу. Всі елементи мають чітке позиціонування в межах сітки, що забезпечує однозначну взаємодію з алгоритмами навігації та логікою обробки замовлень. Таким чином, дана структура є збалансованою за складністю і функціональністю, дозволяє провести ефективне тестування системи, а також, за потреби, масштабуватися чи модифікуватися під нові конфігурації.

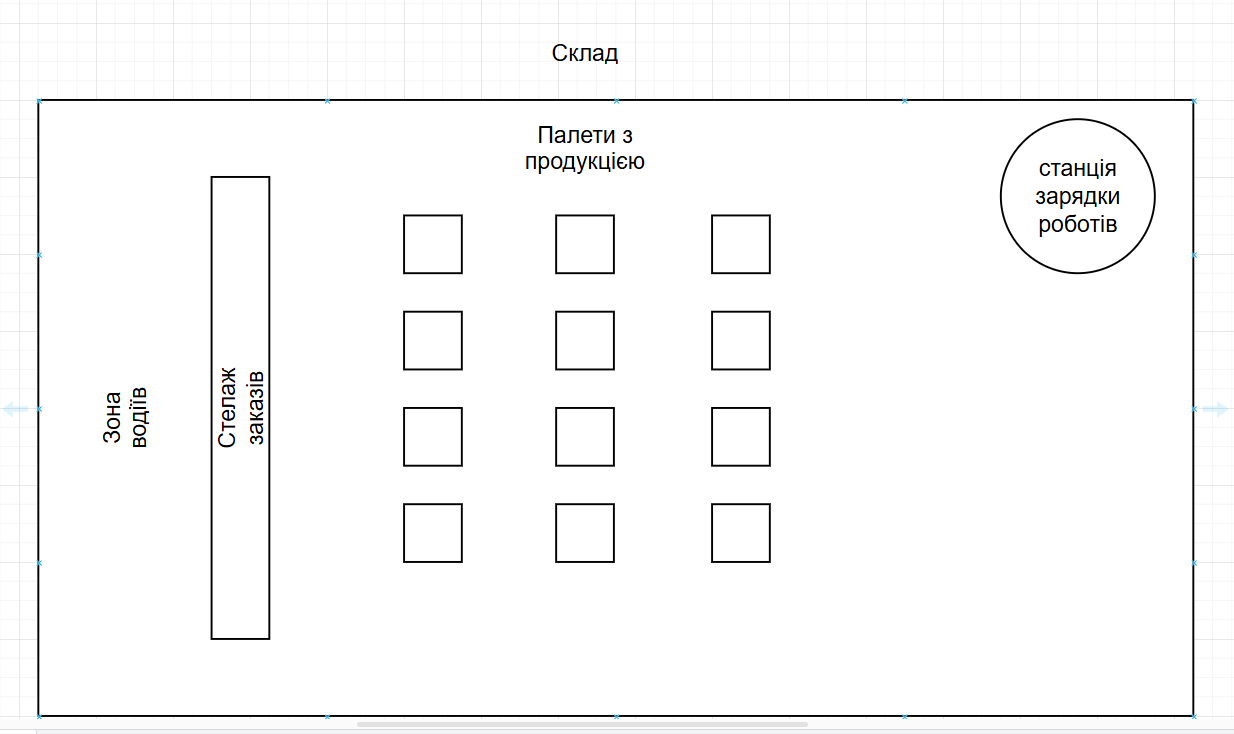


Рисунок 3.3 — Прототип майбутнього складу

## 3.4 Проєктування обробки замовлень

### 3.4.1 Структура замовлення та його компоненти

У межах підсистеми автоматизованого управління складом передбачається реалізація логіки створення та початкової обробки замовлень, яка базується на типовій практиці двокомпонентної моделі: **заголовка замовлення** та **його позицій**. Як було закладено при проєктуванні бази даних, структура зберігається у двох взаємопов’язаних таблицях — orders (основна інформація про замовлення) та order\_items (деталізація за товарними позиціями).

З боку логіки генерації замовлень передбачається модуль, відповідальний за автоматичне створення тестових запитів на доставку товарів. Основна ідея полягає в тому, щоб формувати симульовані сценарії замовлень для перевірки ефективності обробки, навігації та інвентаризації. Передбачається, що під час запуску симуляції система за вказанням тестувальника—адміністратора самойстійно буде генерувати замовлення зі статусом **«pending»,** після чого робот обирає з доступного каталогу один або кілька товарів у випадковому порядку.

Кожна з товарних позицій замовлення повинна містити унікальний ідентифікатор товару, кількість одиниць, а також бути пов’язана з відповідним order\_id. Кількість товару може генеруватися випадково в заданих межах (наприклад, від 1 до 5 одиниць), що дозволяє моделювати як прості, так і комплексні запити.

Генерація відбувається **в рамках транзакції**, щоби гарантувати узгодженість усіх записів у базі даних. Зокрема, після вставки нового запису у таблицю orders, одразу ж отримується його id, який потім використовується для заповнення order\_items. Такий підхід дозволяє точно зв’язати всі елементи одного замовлення, уникнути колізій при паралельному доступі до системи та забезпечити збереження цілісності даних.

Таким чином, у процесі проєктування було визначено, що система повинна підтримувати **автоматизоване створення замовлень**, які за своєю структурою відображають реальні логістичні сценарії. Це дозволяє в подальшому ефективно тестувати поведінку роботів, логіку маршрутизації, обробку помилок і балансування навантаження.

### 3.5.2 Життєвий цикл замовлення

У рамках проєктування системи автоматизованого управління складом важливою складовою є формалізація **життєвого циклу замовлення** тобто визначення всіх можливих станів, у яких може перебувати замовлення, а також правил переходу між ними.

На етапі проєктування передбачається, що кожне замовлення буде проходити через **чітко визначену послідовність станів,** які дозволяють контролювати процес обробки та мінімізувати ризики дублювання, втрати чи некоректного виконання. Для зберігання поточного стану буде використовуватись окреме текстове поле status у таблиці orders.

Очікується, що життєвий цикл замовлення складатиметься з таких основних статусів.

* 1. pending — базовий стан, у якому перебуває щойно створене замовлення. На цьому етапі замовлення ще не обробляється жодним із роботів;
  2. processing — замовлення перейшло на етап активної обробки. Робот закріплений за цим замовленням, починає збір товарів із палет та пошук полиць;
  3. done — усі позиції замовлення успішно зібрані та розміщені на полицях. Замовлення вважається повністю виконаним;
  4. partial — частина товарів була доставлена, але деякі позиції неможливо обробити (наприклад, через відсутність на складі);
  5. failed — жодна позиція не була виконана через критичні помилки, відсутність товарів або збої у роботі.

Під час переходів між цими статусами передбачається використання **механізмів перевірки стану** (тобто перевірка, що статус pending, перш ніж змінити його на processing), що дозволить уникнути одночасної обробки одного й того самого замовлення кількома агентами. Відповідна діаграма представлена на кресленику ІК12.310БАК.006 Д3.

Для забезпечення надійності ці переходи повинні реалізовуватися в рамках **транзакцій**, щоби уникнути станів гонки та непередбачуваних змін.

### 3.5.3 Перевірка доступності товарів

У межах підсистеми обробки замовлень важливо передбачити механізм ефективної перевірки наявності товарів, що входять до складу кожного замовлення. Одним із ключових завдань цього етапу проєктування є визначення логіки пошуку палет, на яких фізично зберігаються необхідні одиниці товару.

Передбачається, що кожне замовлення містить перелік товарів із зазначенням кількості. Для реалізації механізму їх отримання потрібно закласти логіку пошуку серед доступних палет складу. Концептуально цей пошук має здійснюватися за двома основними критеріями:

1. Відповідність товару та необхідної кількості;
2. Просторовий пріоритет вибір найближчої палети з наявністю потрібного товару.

Під час проєктування системи варто також врахувати майбутнє розширення, що передбачає підтримку кількох точок зберігання одного й того самого товару та розробку стратегії вибору оптимального джерела з огляду на логістику руху роботів.

### 3.5.4 Резервування полиць

У процесі обробки замовлення ключовим етапом є визначення місця, куди робот має доставити зібрані товари. Для цього необхідно проєктом передбачити механізм **резервування полиць**, що дозволить уникнути конфліктів між роботами та забезпечити логічне групування замовлень.

#### 3.5.4.1 Правила сумісності товарів на полиці

Під час розміщення товарів на полиці варто дотримуватися принципу сумісності: товари з різних замовлень не повинні зберігатися на одній полиці одночасно. Це дозволить уникнути плутанини та полегшить подальше комплектування й видачу. Також доцільно запровадити можливість пріоритетного використання полиць, на яких вже зберігаються товари того самого замовлення, щоб покращити логічну заповнененість складу.

#### 3.5.4.2 Атомарне резервування

Система резервування повинна базуватись на принципах **атомарності**[[10],](#_Джерела) тобто гарантувати, що полиця не буде призначена кільком роботам одночасно. На етапі проєктування передбачено, що кожна спроба резервування перевірятиме поточний стан полиці перед тим, як надати до неї доступ. Це забезпечить коректність та цілісність даних у багатопоточному середовищі.

#### 3.5.4.3 Зміна статусу полиці

Передбачається, що кожна полиця матиме визначений життєвий цикл у вигляді статусів:

* **free** — полиця вільна і доступна для резервування;
* **reserved** — полиця тимчасово зарезервована під конкретне замовлення або робота, товари ще не доставлені;
* **busy** — полиця заповнена товарами, доступна лише для подальших операцій видачі.

Ця система статусів дозволить координувати доступ до полиць, уникати конфліктів і ефективно використовувати наявний простір.

### 3.5.5. Висновок

Описана підсистема обробки замовлень є однією з ключових складових проєктованої архітектури симуляції роботи складу, адже саме вона забезпечує логічне керування потоками замовлень, від моменту генерації до фінального розміщення товарів на полицях. У рамках цього етапу було закладено концепцію, згідно з якою кожне замовлення має структуроване представлення, проходить чіткий життєвий цикл із фіксованими статусами та перевірками на кожному етапі, а також взаємодіє з системою управління інвентарем через пошук і резервування палет і полиць. Під час проєктування особлива увага приділялася принципам цілісності, атомарності операцій і логістичної сумісності. Передбачено резервування ресурсів, уникнення конфліктів між роботами та забезпечення надійного виконання кожного запиту.

## 3.6 Проєктування навігації та уникнення колізій

### 3.6.1 Архітектура навігаційної підсистеми

На етапі проєктування системи автоматизованого управління складом навігаційна підсистема розглядається як один із ключових компонентів, який забезпечує ефективне пересування мобільних роботів між палетами, полицями та іншими точками складу. Основною концепцією є побудова **багаторівневої модульної архітектури**, яка розділяє відповідальність між окремими функціональними рівнями — навігацією як плануванням шляху та управлінням переміщенням роботів у реальному часі.

Передбачається, що навігаційна підсистема складатиметься з двох основних модулів:

1. **Модуль пошуку шляху (Навігатор)**: Цей компонент відповідатиме за побудову маршруту між початковою точкою та ціллю з урахуванням структури карти складу. Передбачено використання класичних алгоритмів пошуку шляху, таких як A\* та Дейкстри, які забезпечать баланс між точністю та швидкістю. Важливим аспектом є те, що модуль працюватиме незалежно від логіки роботів, приймаючи як параметр спеціальну функцію перевірки доступності клітинки, що дозволяє легко масштабувати рішення та адаптувати його до змін у середовищі;
2. **Модуль управління рухом робота**: Цей компонент відповідатиме за інтеграцію побудованого шляху з поведінкою конкретного робота. Тут будуть реалізовані механізми резервування клітинок, уникнення колізій з іншими агентами, контроль заряду батареї, а також обробка непередбачуваних ситуацій, як—от deadlock або блокування проходу. Для цього проєктом передбачено використання багатопотокової обробки з централізованим контролем доступу до ресурсоємних елементів (наприклад, клітинок, полиць, палет).

Уся архітектура будується з урахуванням принципів **слабкого зв’язування**, що дозволяє окремо оновлювати або змінювати модулі навігації та управління без втручання у загальну логіку системи. Передбачається підтримка гнучкої конфігурації, яка дає змогу адміністратору—тестувальнику обирати бажаний алгоритм пошуку, а також розширювати функціонал за рахунок підключення нових стратегій маршрутизації чи модулів аналізу ефективності.

Таким чином, проєктована архітектура навігаційної підсистеми забезпечить масштабованість, адаптивність та високу надійність роботи в умовах багатокористувацького складського середовища з динамічним змінним станом.

### 3.6.2 Алгоритми планування маршруту

Планування маршруту є фундаментальною складовою навігаційної підсистеми, що визначає ефективність, швидкість і безпечність пересування мобільних роботів складом. На етапі проєктування передбачається реалізація модульного підходу до алгоритмів маршрутизації, що дозволяє як використовувати перевірені класичні методи, так і масштабувати систему шляхом додавання нових стратегій пошуку.

#### 3.6.2.1 Основні вимоги до алгоритмів

У контексті складського середовища основні вимоги до алгоритмів планування маршрутів включають:

1. **Оптимальність шляху** — знаходження маршруту з мінімальними витратами (довжина, час, енергоспоживання);
2. **Швидкість** — низький час обробки запиту, особливо при великій кількості одночасних агентів;
3. **Адаптивність** — можливість швидко реагувати на зміни середовища: зайняті клітинки, рух інших роботів, динамічні перешкоди.

#### 3.6.2.2 Обрані алгоритми

На етапі проєктування передбачається інтеграція двох базових алгоритмів пошуку шляху: A\* (A-star) та Dijkstra.

Алгоритм A\*, це один із найпоширеніших та найефективніших методів пошуку найкоротшого шляху в графі або сітці, що широко застосовується в задачах навігації автономних агентів, таких як мобільні роботи на складі.A\*(A—star) поєднує у собі переваги:

1. забезпечує пошук найкоротшого шляху;
2. **евристичного підходу**, що пришвидшує пошук, орієнтуючись у напрямку цілі.

**Принцип роботи:**

A\* використовує **функцію оцінки вартості шляху** f(n) для кожної клітинки n:

(3.1)

де

* — вартість шляху від початкової клітинки до поточної n;
* — **евристична оцінка** вартості шляху від n до цільової клітинки (heuristic);
* — загальна оцінка вартості маршруту, якщо йти через клітинку n.

В рамках проєктування навігації на дискретній сітці складу, обрано **Евклідову відстань**[[12]](#_Джерела_1) як евристику:

(3.2)

Це забезпечує реалістичну оцінку відстані при русі в будь—якому з 8 напрямків (включаючи діагональні), або Манхеттенську відстань[[13]](#_Джерела_1) (тільки 4 напрямки), якщо рух обмежено:

(3.3)

А тепер вважаю доречним продемонструвати псевдокод алгоритму A\* у вигляді «псевдокоду» оскільки він допоможе краще зрозуміти логіку роботи пошуку шляху на концептуальному рівні. Така візуалізація стане у пригоді при безпосередній розробці модуля навігації та дозволить уникнути помилок під час реалізації ключових етапів алгоритму (див. Додаток Б).

Підсумовуючи інтегрування алгоритму A\* у подальшій реалізації системи навігації повинен забезпечити ефективне та оптимальне прокладання маршрутів між точками на складі. Завдяки використанню евристичної оцінки, A\* дозволяє не лише знаходити найкоротші шляхи, а й робить це швидко, що важливо для системи з багатьма автономними агентами. Такий підхід сприяє зменшенню часу на обробку замовлень, покращенню продуктивності роботів та стабільній роботі в умовах динамічного середовища з великою кількістю об'єктів і потенційних перешкод.

Для того, щоби зробити роботу цікавішою, а пошук шляху адаптивним мною прийняте рішення інтегрувати ще один алгоритм пошуку шляху для подальшого порівняння з основним алгоритмом A\* у контексті швидкості та ефективності. Другим алгоритмом мною був обраний алгоритм Дейкстри, це один із найвідоміших і найстабільніших методів знаходження найкоротшого шляху в графі з ненегативними вагами ребер. Його універсальність і точність робить його ідеальним для задач, де потрібна гарантія знаходження оптимального маршруту, особливо в умовах, коли евристичні припущення (як у A\*) або не підходять, або не дають переваг.

Алгоритм Дейкстри полягає в поступовому розширенні межі від вже відомих найкоротших шляхів до ще невідомих вершин графа. Всі вершини мають значення відстані, яка на початку є нескінченністю (∞), окрім стартової, для якої вона дорівнює нулю. Після цього алгоритм ітеративно оновлює ці значення, обираючи на кожному кроці вершину з найменшою поточною відстанню, і оновлює сусідів, якщо знайдений шлях до них є коротшим за вже відомий. [[14]](#_Джерела_1)

Формально, для кожної вершини *d(v)* обчислюється значення найкоротша відстань від початкової вершини. Після завершення роботи алгоритму кожна вершина матиме значення *ɖ(v),* яке представляє довжину найкоротшого шляху від *s* до *v*, а також попередник у цьому шляху.

Основні формули:

* Нехай: *Q* — це множина всіх ще не оброблених вершин,
* *d(u) —* мінімальна знайдена на поточний момент відстань до вершини *v,*
* *w(u,v) –* вага ребра між вершинами *u,v.*
* Тоді, для кожного сусіда *v* вершини *u*, перевіряється умова оновлення:

(3.4)

Таке оновлення називається relaxation (релаксація ребра), і воно гарантує, що ми не втратимо коротший шлях, якщо він знайдеться пізніше. «Псевдокод» даного алгоритму також наданий у додатку Б.

У контексті проєктованої системи навігації складських роботів алгоритм Дейкстри може бути корисним як альтернативний метод для побудови маршрутів у ситуаціях, де точність і стабільність важливіші за швидкість. Наприклад, у випадках з обмеженим розміром сітки або при аналізі повного розподілу маршрутів на глобальному рівні. На відміну від A\*, Дейкстра не використовує евристику і гарантує абсолютно оптимальний шлях — це робить його доцільним у складних ситуаціях, де не можна покладатися на приблизні оцінки відстаней.

Таким чином, включення цього алгоритму у підсистему навігації дозволить у майбутньому адаптивно перемикатись між стратегіями пошуку шляху залежно від завдань, цілей і поточних умов навігаційного простору.

### 3.6.3 Виявлення та уникнення колізій

Під час проєктування навігаційної підсистеми для автономних мобільних роботів особливу увагу слід приділити виявленню та уникненню колізій у спільному просторі. Роботи, що одночасно переміщуються на складі, можуть створювати ситуації конфлікту, як зоображено на рисунку 3.4, за ресурси середовища: клітинки, зони дій, вузькі проходи тощо. Уникнення таких конфліктів є критично важливим для ефективного та безпечного виконання замовлень.

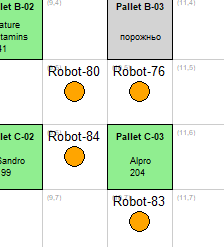


Рисунок 3.4 — Можлива ситуація виникнення конфлікту(стрілка це напрямок руху)

На концептуальному рівні передбачається впровадження системи динамічного резервування наступної клітинки. Кожен робот, отримуючи свій маршрут, повинен «бронювати» відповідні ділянки простору на час переміщення. Завдяки цьому інші агенти системи можуть враховувати зайнятість певних клітинок і планувати свій маршрут відповідно до актуального стану середовища. Такий підхід сприятиме запобіганню одночасному доступу до однієї й тієї ж точки на карті.

У структурі системи має бути закладено механізм моніторингу позицій і намірів роботів. Це означає, що кожен агент системи не лише фіксує своє поточне положення, а й публікує інформацію про свої наміри — наприклад, цільову позицію або наступну клітинку руху. Завдяки такій прозорості взаємодії всі агенти системи можуть приймати зважені рішення з урахуванням присутності інших.

Також важливо передбачити правила пріоритетності в разі потенційного конфлікту. Наприклад, робот, який уже перебуває в русі або ближче до цілі, може отримати перевагу, тоді як інші повинні тимчасово зупинитись або змінити маршрут. У ситуаціях, де взаємне блокування неминуче (наприклад, у вузьких проходах), повинна бути запроєктована стратегія виявлення циклічних залежностей і відповідного реагування: один з роботів здійснює відступ або перебудовує маршрут альтернативним шляхом.

Окремо має бути закладено концепцію очікування або перепланування, яка активується, якщо поточна траєкторія робота перекрита. Замість негайного переміщення робот може обрати варіант тимчасового очікування, або ж звернутись до модуля повторного проєктування маршруту, який врахує нові обставини.

Важливо, що всі рішення щодо уникнення колізій мають прийматись у режимі реального часу, з урахуванням багатопоточності, що властива системам, де функціонують десятки роботів одночасно. У зв’язку з цим під час проєктування передбачається використання **м**еханізмів синхронізації доступу до просторових ресурсів, які забезпечують атомарність дій окремих агентів.

Таким чином, проєктування підсистеми уникнення колізій ґрунтується на наступних концептуальних приципів:

1. динамічне резервування просторових ресурсів;
2. прозора публікація намірів агентів;
3. правила пріоритетності та відступу;
4. механізми виявлення циклічних блокувань;
5. логіка очікування або адаптивного перепланування;
6. гарантії атомарного доступу до критичних ділянок середовища.

Реалізація цих принципів дозволить створити гнучку, надійну систему, здатну до ефективної взаємодії численних роботів у динамічному середовищі складу. Відповідну пропоновану реалізацію, а саме діаграму виявлення та вирішення колізій, надано у кресленнику(ІК12.310БАК.006 Д4)

### 3.6.5 Адаптивна навігація та обхідні шляхи

У складських умовах, де одночасно діє велика кількість мобільних роботів, виникають ситуації тимчасової недоступності окремих ділянок простору, наприклад, через зайнятість клітинки іншим роботом, фізичну перешкоду чи заблокований прохід. Тому під час проєктування навігаційної підсистеми важливо передбачити адаптивну поведінку роботів, яка дозволяє не лише знаходити базовий маршрут до цілі, але й гнучко реагувати на динамічні зміни навколишнього середовища.

#### 3.6.5.1 Пошук альтернативних позицій підходу

Однією з ключових концепцій є можливість пошуку альтернативних точок підходу до цілі. У випадках, коли кінцева позиція (наприклад, задана полиця) тимчасово недоступна, робот не повинен зупинятись у режимі очікування перед перешкодою. Натомість у проєктній логіці закладається, що робот може визначити набір суміжних допустимих клітинок, які розташовані поряд із ціллю, і намагатись дістатися до однієї з них. Це дозволить зменшити час простою та уникнути скупчення роботів в одному місці.

#### 3.6.5.2 Динамічне очікування звільнення позицій

Інша концепція, це контрольоване очікування. Якщо навіть альтернативні позиції виявляються зайнятими, робот може увійти в режим динамічного очікування, тобто призупинити рух на певний короткий проміжок часу, регулярно перевіряючи, чи звільнилася потрібна позиція. Такий підхід ефективний у ситуаціях, коли перешкода є тимчасовою, наприклад, інший робот завершує своє завдання і скоро звільнить клітинку. Передбачається, що очікування буде обмеженим за часом (тайм—аут), щоб уникнути нескінченного блокування.

#### 3.6.5.3. Випробування обхідних маршрутів з обмеженим часом

Якщо очікування не дало результату, система має підтримувати спробу побудови обхідного маршруту. Це означає, що навігаційний модуль ініціює пошук нового шляху до цілі або до найближчої альтернативної точки, який не проходить через заблоковану клітинку. Водночас на етапі проєктування слід закласти тимчасове обмеження: якщо пошук нового шляху перевищує допустимий поріг часу (встановлений для підтримки загальної ефективності системи), робот переходить у режим очікування або надсилає запит на розв’язання конфлікту вищому рівню управління.

Уся логіка адаптивної навігації має працювати в режимі реального часу з урахуванням багатопоточності, де кожен робот автономно приймає рішення на основі локальної інформації та глобальних параметрів середовища. При цьому важливо, щоб проєктовані механізми:

1. дозволяли швидко змінювати маршрут у разі появи нових перешкод;
2. уникали глобального перенавантаження системи через нескінченне перепланування;
3. забезпечували стійкість навігації в умовах високої динаміки складу.

Завдяки впровадженню описаних концепцій, система зможе гарантувати не лише мінімальний час виконання завдань, а й високу гнучкість у реагуванні на непередбачувані ситуації, що є критично важливим для реальних умов експлуатації.

## 3.7. Проєктування логіки зарядки

У процесі експлуатації автономних мобільних роботів критичним є питання забезпечення **енергетичної стійкості** системи. Тобто, необхідно гарантувати, що кожен робот здатен завершити своє завдання без повної втрати заряду та має чітку стратегію повернення до зарядної станції. Тому в межах цього підпункту розробляється **концептуальна архітектура підсистеми зарядки,** яка забезпечить **безперервність обслуговування замовлень** навіть за великої кількості роботів у спільному середовищі.

### 3.7.1 Принципи роботи енергетичного моніторингу

У кожного робота має бути вбудований віртуальний сенсор енергії, який фіксує:

1. Поточний рівень заряду акумулятора (%);
2. Витрати енергії на переміщення (в умовних одиницях на 1 клітинку);
3. Прогнозований енергетичний бюджет для обслуговування нового замовлення.

Ці параметри дозволяють системі в режимі реального часу приймати рішення, чи варто починати нове завдання, або необхідно вирушити до найближчої зарядної станції.

### 3.7.2 Оцінка доцільності виконання замовлення

Основним проєктним рішенням у цьому контексті є створення інтелектуального механізму моніторингу енергетичного стану кожного робота. Кожен пристрій повинен мати змогу не лише зчитувати власний рівень заряду акумулятора, а й прогнозувати витрати енергії, пов’язані з поточним або потенційним маршрутом. Ці дані мають враховуватись ще до початку виконання нового замовлення. У випадку, якщо очікуване споживання перевищує наявний заряд з урахуванням резерву на аварійне повернення, робот повинен не розпочинати завдання, а натомість самостійно зініціювати перехід до найближчої зарядної станції.

Важливою частиною проєктування є визначення способу обчислення “енергетичної вартості” завдання. На цьому етапі закладається концепція, згідно з якою маршрут оцінюється не лише з точки зору відстані, а й з погляду енерговитрат. Орієнтовна формула, яка проєктується для реалізації на практиці, має вигляд:

**Х = E\_наявна\_эмність – (E\_шлях\_до\_палети + E\_шлях\_до\_полиці +** +**E\_шлях\_назад) (3.5)**

де кожна складова відповідає енергії, потрібній для підходу до палети, доставки товару, повернення до зарядки. На виході отримуємо процент заряду(х) батареї, яку буде мати робот пілся виконання завдання. Така модель дозволяє вбудувати енергетичну передбачливість у саму логіку ухвалення рішень, що значно підвищує автономність і безпечність системи.

Таким чином, уся логіка зарядки проєктується як **автономна та адаптивна**, інтегрована у загальну архітектуру взаємодії роботів. Особливістю запропонованої моделі є те, що кожен робот має закріплену **індивідуальну зарядну станцію**, що дозволяє повністю уникнути ситуацій черги, конфліктів за доступ до енергоресурсів та спрощує процес навігації до точки підзарядки. Завдяки цьому, кожен робот у будь—який момент може автономно повернутись до власної станції, не перетинаючись з іншими агентами. Це рішення суттєво спрощує логіку планування маршрутів до зарядки, усуває потребу в централізованому диспетчерському розподілі доступу та підвищує надійність системи в цілому. Такий підхід дозволяє структурувати процес зарядки як індивідуальний етап життєвого циклу кожного робота, з чітким енергетичним контролем, передбачуваністю та повною відсутністю конкуренції за ресурси.

## 3.8 Висновки по розділу проєктування

У межах етапу проєктування було сформовано узгоджену архітектурну концепцію інтелектуальної складської системи, що базується на координації мобільних роботів, оптимізації логістичних потоків та ефективному управлінні ресурсами. Спроєктовано логічну структуру складського простору, включаючи розміщення палет, полиць і зарядних станцій, з урахуванням потреб навігації, розділення потоків та уникнення колізій.

Опрацьовано концепцію життєвого циклу замовлення з урахуванням атомарного резервування ресурсів, контролю статусів і забезпечення логічної узгодженості між діями роботів. Запропонована навігаційна підсистема охоплює два підходи до пошуку шляху, евристичний (A\*) та класичний (Dijkstra), а також моделі обробки колізій, динамічного обходу перешкод і адаптивного реагування на зміни в середовищі.

Концепція індивідуальних зарядних станцій для кожного робота дозволяє забезпечити безконфліктну організацію процесу підзарядки та уникнути вузьких місць у системі обслуговування енергетичних потреб. Передбачено механізми оцінки енергетичних витрат при плануванні маршрутів, що посилює загальну адаптивність системи.

Усі функціональні модулі проєктувалися з урахуванням майбутньої масштабованості, модульності та подальшої інтеграції. Результати проєктування формують міцну основу для наступного етапу розробки, забезпечуючи структурну цілісність системи, логічну завершеність концепцій і готовність до переходу до практичної реалізації.

# **РОЗДІЛ 4.**

# **РЕАЛІЗАЦІЯ**

## 4.1 Архітектура програмної реалізації

### 4.1.1 Вибір технологій

Для реалізації підсистеми обраний наступний стек технологій:

1. Основна Мова программування: Python 3.12;
2. База даних: Microsoft SQL Server;
3. Робота з БД: Бібліотека pyodbc для підключення та виконання SQL-запитів;
4. Графічний інтерфейс: Tkinter (стандартна бібліотека Python для адмін панелі);
5. Додаткові модулі: json, os, datetime, threading, random, math, time, heapq

### 4.1.2 3агальний опис реалізації архітектури підсистеми

На основі спроєктованої концепції структури проєкту у розділі проєктування, було реалізовано модульну архітектуру програмної системи управління автономними складськими роботами. Реалізація поділена на три основні компоненти: logic/ (логіка роботи роботів), simulation/ (модулі симуляції та статистики), та db/ (взаємодія з реляційною базою даних). Центральними точками входу виступають файли main.py та test.py, що дозволяють запускати систему в робочому чи тестовому режимі відповідно. Поділ системи на окремі модулі та компоненти є ключовим принципом сучасного проєктування, що дозволяє досягти високої гнучкості, повторного використання коду, зручності тестування та масштабованості. Завдяки модульному підходу кожен компонент системи виконує чітко визначену функцію, що значно спрощує його відлагодження, оновлення та розширення. Наприклад, навігаційний модуль можна вдосконалити або замінити, не зачіпаючи логіку роботів, а статистику можна збирати та аналізувати незалежно від основного виконання симуляції. Така архітектура не лише підвищує надійність і стабільність системи в умовах зміни вимог, а й дозволяє ефективно організувати командну розробку, де кожен учасник може працювати над своїм компонентом, не створюючи конфліктів із рештою функціоналу. У результаті система стає більш прозорою, розширюваною та стійкою до змін, як у контексті технічних рішень, так і в умовах зростання складності або навантаження. Ключові модулі:

* 1. robot.py — основний клас RobotNavigator, що інкапсулює логіку автономної поведінки роботів: від навігації та управління батареєю до обробки замовлень та взаємодії з базою даних. Реалізовані механізми уникнення колізій, резервування клітинок, виявлення deadlock—ситуацій та відновлення після помилок;
  2. navigator.py — основний класс Nabigator, незалежний модуль навігації, що реалізує алгоритми A\* та Dijkstra з можливістю автоматичного вибору оптимального варіанту на основі статистики. Навігація відбувається з урахуванням зайнятих клітинок і змін у середовищі;
  3. orders.py — модуль управління замовленнями, який генерує, розподіляє і оновлює замовлення, синхронізуючи це з базою даних. Підтримується повний життєвий цикл: від pending до done;
  4. file\_diagnostics.py — компонент статистики: збір метрик за кожним роботом, алгоритмом, замовленням. Генерує текстові звіти у (results\_of\_simulation.txt) для аналітики;
  5. warehouse\_map.py — описує сітку складу, розташування полиць, палет, зарядних станцій. Служить базою для побудови маршрутів та візуалізації;
  6. connection.py забезпечує потокобезпечне підключення до MS SQL Server. Кожен робот має окреме з'єднання, що виключає конфлікти під час паралельної роботи.

### 4.1.3. Принципи модульного проєктування та розподілу відповідальності:

1. Слабкий зв'язок між компонентами: Модулі не мають жорстких залежностей між собою. Наприклад, Navigator передається до RobotNavigator через композицію. База даних доступна через get\_connection(). Статистика збирається незалежно від логіки руху;
2. Масштабованість**:** Система підтримує динамічне створення до 15 роботів з унікальними Id через start\_robot(robot\_id). Роботи функціонують у власних потоках, використовуючи синхронізовані глобальні структури reserved\_cells, robot\_destinations;
3. Потокобезпечність**:** Використовуються Lock() — блокування для захисту загальних структур. Кожен робот має status\_lock, planned\_path\_lock для контролю свого стану;
4. Відмовостійкість**:** Реалізовано повторні спроби при помилках навігації (safe\_move\_to()), fallback—позиції для тупикових ситуацій, обробка замовлень без наявності всіх товарів, автоматичний перехід на зарядку при низькому заряді.

Підсумовуючи, уся система побудована, як симуляційна платформа для дослідження логістики на основі інтелектуальних агентів. Вона дозволяє моделювати реальні сценарії складу, тестувати різні алгоритми та оперативно адаптуватися під змінні умови без переписування основної логіки

## 4.2 Реалізація цифрової сітки складу та адмін - панелі

Першою важливою задачею проєкту стала реалізація цифрової сітки складу, оскільки саме вона формує просторову основу для всієї логіки навігації, розміщення об’єктів та взаємодії роботів з навколишнім середовищем. Без чіткої структури координат було б неможливо реалізувати алгоритми пошуку шляху, резервування клітинок чи розміщення палет і полиць у відповідних позиціях. Цифрова сітка дозволяє задати топологію складу у вигляді зручної для обчислень матриці, забезпечити однозначну ідентифікацію кожної клітинки та ефективно синхронізувати всі модулі системи, від логіки роботів до візуалізації. Саме тому її побудова стала фундаментальним кроком для забезпечення узгодженості даних та стабільної роботи всієї інфраструктури.

Цифрова модель складу реалізована у вигляді координатної сітки розміром 20×41, що імітує фізичне середовище логістичного простору, включаючи палети, полиці, вільні зони для переміщення, а також зарядні станції(зона з (19.2) по (19.16)). Ініціалізація цієї сітки виконується у модулі warehouse\_map.py, де визначено координати всіх ключових об’єктів, shelf\_coords, pallet\_coords та charging\_station\_coords, що потім використовуються іншими модулями для планування маршрутів, резервування клітинок і візуалізації.

Ключовим інтерфейсом взаємодії з цифровою сіткою виступає графічна **адміністративна панель**, реалізована у модулі admin\_panel\_gui.py. Вона об'єднує функціонал керування замовленнями, моніторингу полиць, а також динамічної візуалізації всього складу. Основною функцією цього модулю є «def draw\_warehouse()», ця функція надає повноцінне графічне представлення адмін панелі з використанням бібліотеки tkinter. Інтерактивна панель дозволяє не лише переглядати поточний стан складу, але й прямо взаємодіяти з логікою обробки замовлень. Наприклад, користувач може виділити конкретне замовлення у вкладці «Замовлення», або натиснути кнопку «Обробити», і система автоматично обробить замовлення(Рисунок 4.1). Після виконання можна переглянути статус полиць у вкладці «Полиці», де відображається інформація про замовлення, яке закріплене за кожною полицею(Рисунок 4.2).

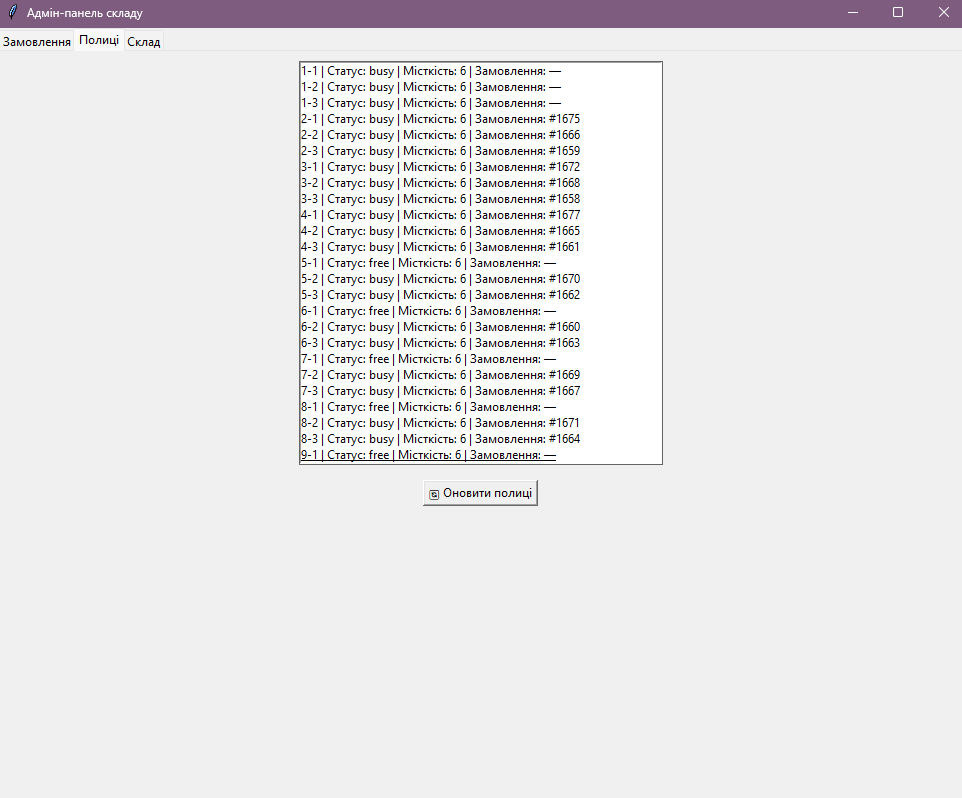
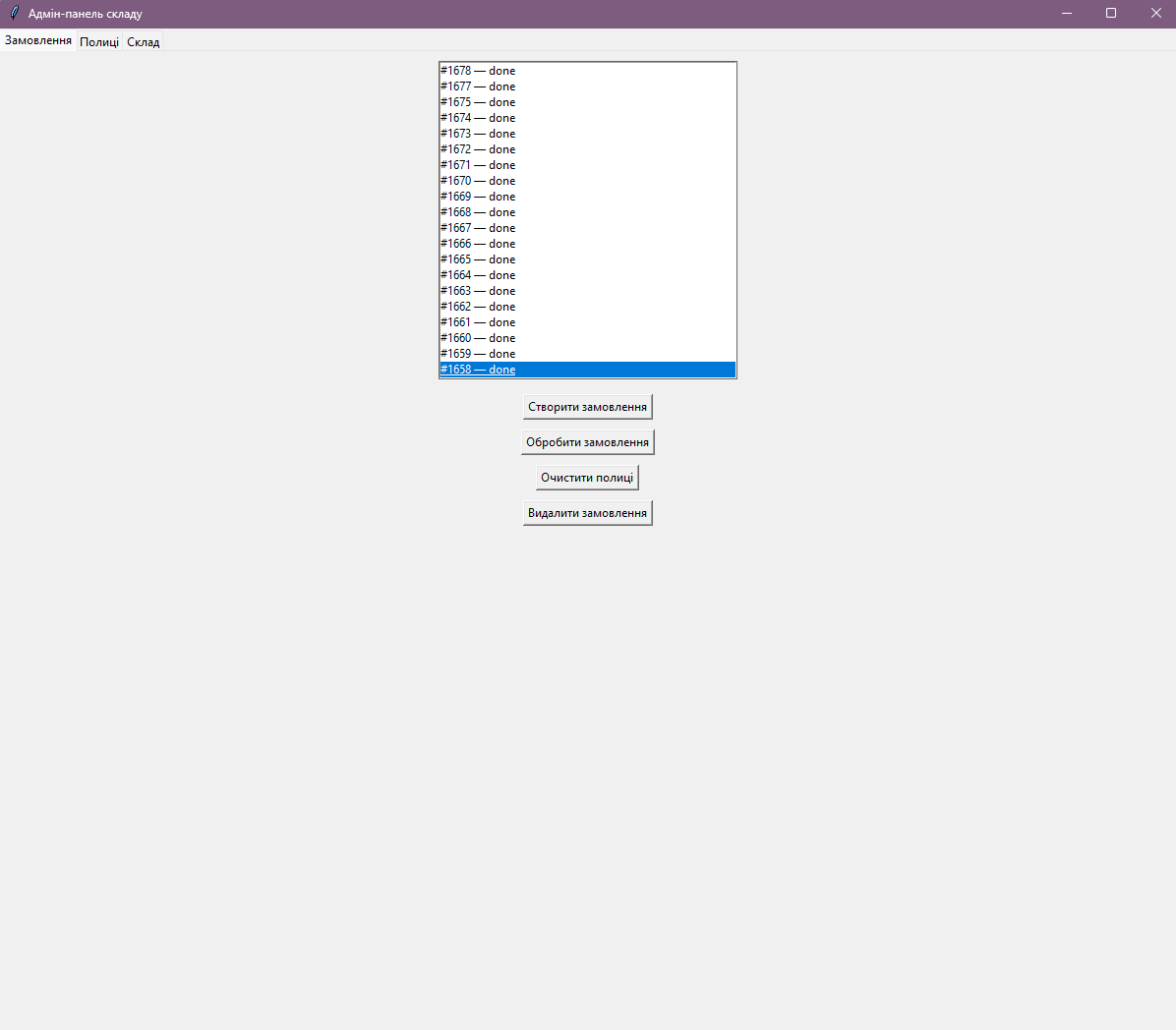


Рисунок 4.1 — Вкладка Замовлення Рисунок 4.2 — Вкладка Полиці

У графічному представленні кожна клітинка сітки відповідає фізичній координаті на складі. Зеленим кольором позначаються палети з товарами, червоним — зайняті полиці, блакитним — вільні полиці, сірим — порожні палети, а синім — зарядні станції. Роботи позначаються помаранчевими кружками з підписами типу Robot:Id, і розміщуються відповідно до своїх координат у базі даних. Завдяки функції auto\_update\_warehouse() панель оновлює стан візуалізації кожні 0.5 секунди, забезпечуючи актуальну картину складу під час симуляції(Рисунок 4.3).

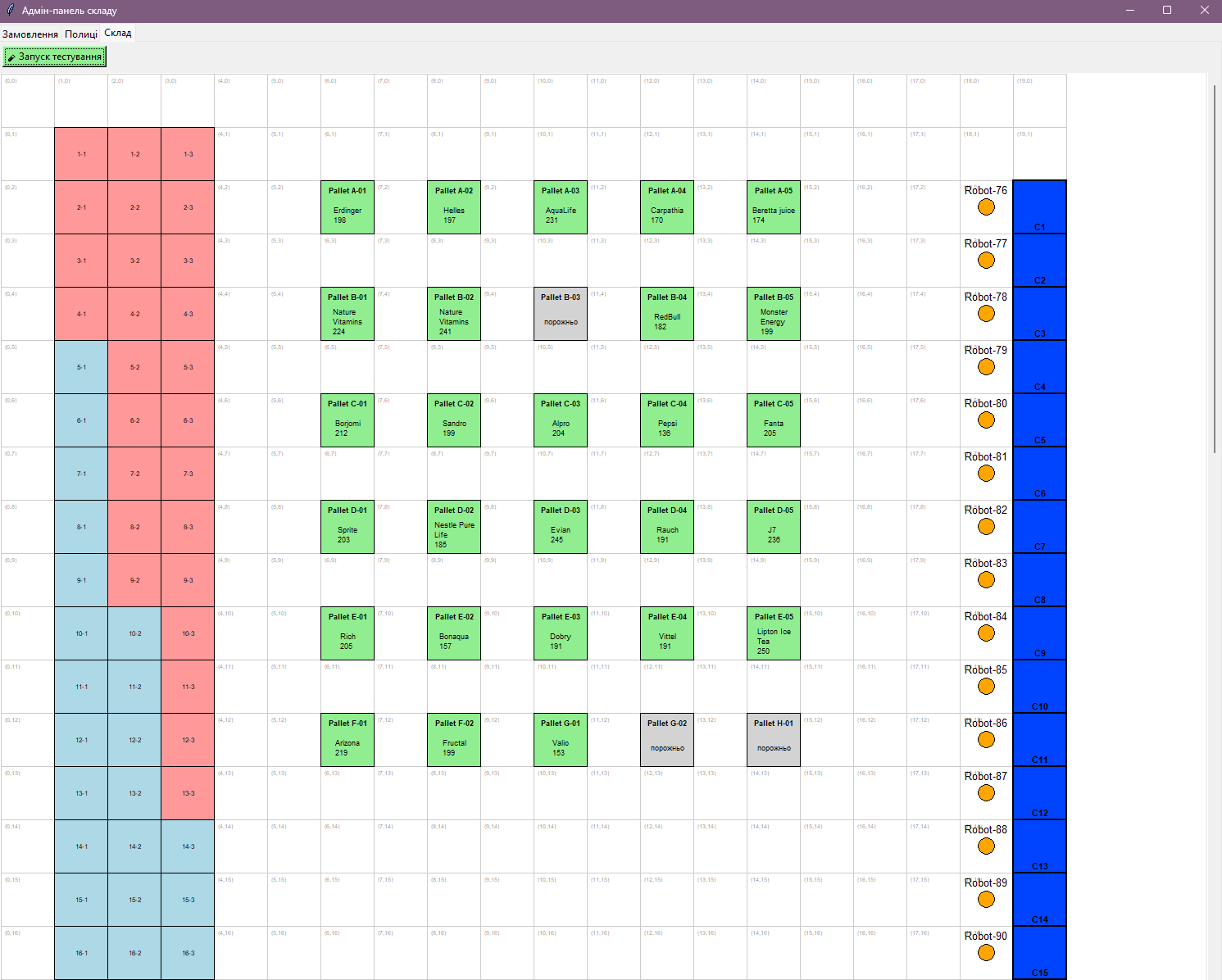
****

Рисунок 4.3 — Цифровова сітка складу

Таким чином, реалізована цифрова сітка складу не лише формально моделює структуру середовища для алгоритмів навігації, а й надає потужний візуальний та інтерактивний інструмент контролю і налагодження через адміністраторську панель. Це забезпечує зручність тестування, та дозволяє ефективно відслідковувати дії кожного робота в реальному часі.

## 4.3 Механізм створення симуляційного замовлення

В рамках модулю orders.py було реалізовано механізм генерації симуляційних замовлень, який дозволяє зімітувати роботу реального складу. Цей механізм є ключовим для тестування поведінки роботів, перевірки алгоритмів навігації та загального функціонування логістичної системи. Замовлення створюються автоматично через виклик функції generate\_random\_order(conn),(див. Додаток А) в адміністративної панелі шляхом натискання кнопки «Створити замовлення».

Під час генерації нового замовлення система випадковим чином обирає кілька палет з наявного інвентарю складу, після чого формує набір товарів із зазначенням кількості для кожного. На цьому етапі не визначається цільова полиця, вона не закріплюється за товаром наперед. Усі замовлення зберігаються в таблиці orders, а їх позиції у таблиці order\_items, де фіксується Id товару, кількість, а також посилання на палету, з якої товар має бути взятий. Відповідна діаграма демонструє цей процес у кресленику ІК12.310БАК.006 Д5.

Реальне розміщення товару на полиці відбувається не під час створення замовлення, а безпосередньо в момент його обробки. Якщо обробка виконується у ручному режимі, то користувач натискає кнопку «Обробити замовлення» в інтерфейсі адміністративної панелі, після чого запускається відповідний процес. У разі реальної симуляції, цим займається робот, який під час виконання завдання самостійно шукає вільну сумісну полицю, використовуючи метод find\_free\_shelf(). Таким чином, вибір полиці для доставки є динамічним і залежить від фактичного стану складу на момент виконання замовлення.

Кожне замовлення має чітко визначений життєвий цикл зі статусами. Початково замовлення має статус pending, що означає його готовність до обробки. Коли виконується фізичне переміщення товарів роботами, статус змінюється на processing. Після завершення всіх дій замовлення переходить у статус done, а у випадку часткового виконання чи проблем —відповідно в partial або failed, якщо товар не був доставлен до полиці. Це допомогає адміністратору розуміти поточний та фактичний статус замовлень.

Структура одного замовлення складається з запису у таблиці orders, що містить унікальний ідентифікатор, статус і дату створення, а також відповідного набору записів у таблиці order\_items, які включають ідентифікатори товарів, де ці товари наразі зберігаються. Система не прив'язує ці товари до конкретних полиць до моменту фактичного виконання завдання, що дозволяє адаптувати процес до реального стану складу в кожен конкретний момент часу.

Такий підхід забезпечує гнучкість у формуванні логістичних сценаріїв і дозволяє точно відтворювати умови динамічного складу, де рішення про розміщення та маршрут приймаються в реальному часі з урахуванням актуального стану інфраструктури.

## 4.4 Логіка взаємодії робота з замовленням

4.4.1 Опис фіксованих умов та обмежень системи

Перш ніж перейти до опису самої логіки призначення й обробки замовлення, варто сформулювати ті жорсткі (фіксовані та незмінні в часі) умови й обмеження, які закладені в реалізації моїй системи. Саме вони задають рамки, у межах яких працюють усі алгоритми навігації, планування, резервування та обробки замовлень. Ось як це реалізовано:

Почнемо з того, що у системі жорстко задане, що кожен робот має обмеження на кількість позицій товару, які може одночасно перевозити. Це параметр max\_capacity = 6. Іншими словами, якщо замовленння вимагає більше ніж шість одиниць якогось конкретного товару, робот не зможе забрати ці всі позиції одним підходом до палети. У такій ситуації він забере рівно шість одиниць (а якщо в замовленні залишилося менше, ніж шість, тоді всю цю залишкову кількість). Тільки після доставки цих шести одиниць до призначеної полиці, звільнення місця в кузові та тимчасового оновлення власної внутрішньої пам’яті робот може повернутися й продовжити збір із тієї ж чи наступної палети, якщо залишилися невиконані позиції. Таке обмеження дозволяє імітувати фізичні обмеження ємності контейнера, а також гарантує, що жоден робот не спробує в якийсь момент узяти з палети, скажімо, двадцять одиниць, переклавши це все в один підхід і при цьому «забивши» інші важливі логістичні зв’язки.

Батарея ще один ключовий ресурс із жорсткими порогами. Сама величина заряду (від 0 % до 100 %) змінюється в залежності від руху, у коді визначено незмінний пороговий рівень: battery\_threshold = 15%

Тому, якщо перед початком обробки нового замовлення (йдеться про поїздку до палети, потім до полиці й зворотний шлях до стандартної позиції) робот самостійно переконується, що навіть із витратами на весь маршрут залишок заряду не опуститься нижче 15 %, він приступає до виконання. Якщо ж оцінка показує, що після виконання цього замовлення допустимий мінімум 15 % не збережеться, робота не бере замовлення. У коді це відбувається через метод can\_complete\_order\_with\_battery(), де враховується фактична вартість руху до палети (метод calculate\_path\_battery\_cost) і приблизні (з «коефіцієнтом безпеки») витрати переходу до полиці та повернення. Якщо результатом такого підрахунку є число менше ніж 15, робот одразу викликає go\_to\_charging\_station() без жодної спроби набирати товар із палети. За рахунок цієї перевірки, ніколи не буде ситуації, коли під час перевезення замовлення робот раптово розрядиться «посередині шляху» і не зможе ні рухатися далі, ні передати вантаж. Відповідний код можно переглянути у додатку А.

Не менш важлива жорстка умова стосується підходів до полиць і палет. У коді для підходу до полиць фіксується позиція з абсцисою 4 (координата (4, y)), де y відповідає найближчий точці до потрібної полиці. Це означає, що робот може покласти товар на полицю з найближчої точки. В обох методах (find\_approach\_position\_for\_pallet\_improved і get\_approach\_position\_improved) є однакова незмінна стратегія: спершу перевірити чотири «основні» сусідні точки за допомогою DIRECTIONS\_4. Якщо хоча б одна вільна, повернути її. Якщо всі зайняті, дочекатися їх звільнення максимум 30 секунд (max\_wait\_time = 30). Якщо й після цього часу нічого не звільнилося, викликати find\_alternative\_approach\_positions, яка сканує радіуси 1, 2, 3 навколо цілі, збираючи всі вільні точки, сортує їх за евклідовою відстанню до поточної позиції робота й повертає список із 3–5 найближчих. Це жорсткі правила, які дозволяють уникнути безкінечного очікування. Якщо альтернативних не знаходиться, в обох алгоритмах повертається None.

Ну і кінцевою умовою є та сама стратегія паралелізму, яка описувалась в проектуванні: у функції run\_robot(robot\_id, ...) одразу створюється окремий потік Thread(target=robot.run) для кожного екземпляру RobotNavigator. Якщо в системі, наприклад, модель передбачає, що на складі одночасно працює 10 роботів, саме 10 об’єктів RobotNavigator (із індивідуальними robot\_id) живуть індивідуальними потоками. Цей підхід « один потік — один робот « є однозначним і не змінюється в процесі. У нас немає пулу завдань, де потоки могли б динамічно перерозподілятися; як тільки створився «поток—робот», він працює, доки запущений процес.

### 4.4.2 Призначення замовлення (вибір, перевірка, закріплення)

На першому етапі взаємодії з замовленням кожен автономний робот повинен самостійно виявити нове доступне завдання, перевірити можливість його виконання за поточних умов та закріпити це замовлення за собою, щоби уникнути конфліктів із іншими роботами. Ця процедура є детермінованою, атомарною та виконується в окремому потоці для кожного робота.

Вибір замовлення починається з пошуку першого доступного запису у таблиці orders, що має статус 'pending'. Робот надсилає SQL—запит з обмеженням Top 1(запит поверне тільки перший рядок із результату) і сортуванням за Id, таким чином вибирається замовлення, яке надійшло першим. У разі, якщо жодне замовлення не знайдено, робот повертається у стан очікування або виконує перехід до зарядної станції (за умовами низького заряду).

Якщо замовлення знайдено, робот отримує order\_id і формує запит до таблиці order\_items, щоб отримати повний перелік товарних позицій, що входять у це замовлення. Цей список формується у вигляді масиву пар (item\_id, quantity) і зберігається для подальшої перевірки.

Після цього робот виконує попередню валідацію: перевіряє, чи доступна хоча б одна одиниця кожного товару на палетах складу, і чи достатній поточний рівень заряду батареї для того, щоб зібрати й доставити хоча б одну позицію. У разі, якщо палети з потрібними товарами не знайдено, робот позначає замовлення як failed, якщо частково — то після доставки до полиці товар позначається як partial. А якщо рівень батареї занадто низький, робот пропускає це замовленння(це замовлення обробить інший робот) та переходить на зарядку.

Лише після успішного проходження цієї перевірки робот закріплює замовлення за собою, змінюючи його статус у базі даних з 'pending' на 'processing'. Ця операція відбувається атомарно за допомогою SQL—запиту з умовою WHERE status = 'pending'. Таким чином, інші роботи, які паралельно могли отримати те саме order\_id, більше не зможуть його обробити.

Такий підхід гарантує принцип «first come, first served» — перший робот, який виконав успішну транзакцію, отримує повні права на виконання замовлення. Всі подальші дії з обробки замовлення відбуваються вже в контексті цього робота, з повною відповідальністю за збір і доставку товарів.

Цей механізм забезпечує узгоджене середовище для розподілу замовлень між роботами, запобігає одночасному доступу до одного й того самого ресурсу та є ключовим для надійності багатопоточної системи обробки.

### 4.4.3 Взаємодія з інвентарем (палети, товари)

Коли батарея достатня і хоча б один із товарів на побачених палетах є доступним, сам робот маркірує у базі замовлення як «processing», щоб наступні роботи не намагалися брати це саме замовлення паралельно. Наступною логікою є перевірка доступності кожного товару (чи немає його у потрібній кількості хоча б на одній палеті), це дозволяє роботам пропускати ті позиції, яких тимчасово не існує на складах, і не блокувати замовлення повністю: якщо певний item\_id не знайдений на жодній палеті, робот просто повідомляє про це в консолі, додає ідентифікатор цієї палети (якщо він хоча б раз трапився у невдалій спробі) у множину failed\_pallets, і переходить до наступного товару. При цьому, якщо всі товари з all\_order\_items не доступні, замовлення одразу помічається як «failed» і повертається False.

Далі стартує контур, у якому робот обробляє по черзі всі позиції, що є у масиві available\_items(доступні товари), і саме тут вступає у гру метод process\_order\_item\_improved(обробка позиції з замовлення).Цей метод складається з ітераційного підходу: доки залишок необхідного товару більше нуля, робот повинен зібрати всі позиції з замовлення. Спочатку для даної позиції товара робот повторно шукає найближчу палету з цим товаром, виключаючи ті палети, які не вдалися раніше (використовується множина failed\_pallets). Якщо після виключення таких палет жодна не підходить, робот чекає кілька секунд (time.sleep(5)) і, оновивши retry\_count(кількість спроб), робить ще одну спробу. Коли знаходиться палета її позиція передається до методу find\_approach\_position\_for\_pallet (знайти точку підхода до палети). Цей допоміжний метод визначає точку, до якої робот повинен підійти безпосередньо до палети, враховуючи можливі зайнятість сусідніх клітинок. Якщо жодна з «прямих» сусідніх клітин не вільна, робот протягом обмеженого часу (до 30 сек) чекає, доки хоча б одна з них звільниться; якщо й тоді нічого не вдалось, він формує список альтернативних позицій у межах радіуса 3 клітин навколо палети і повертає першу (найближчу) з них. Якщо жодна альтернатива не знаходиться, це значить підхід до палети наразі заблокований, палета позначається як невдала (failed\_pallets.add(pallet\_id)) і до retry\_count(кількість спроб) додається одиниця. У разі, якщо find\_approach\_position\_for\_pallet повертає None, робот одразу перекидає обробку на наступну ітерацію, у якій будуть або інші палети з потрібного списку, або коли ресурси неочікувано змінились буде зроблена повна повторна спроба з тими ж палетами через п’ять секунд.

Після цього робот виконує фільтрацію: із наявних палет вибирається найближча за евклідовою відстанню до поточного положення. Потім виконується виклик функції пошуку шляху до палети, із врахуванням зайнятих клітинок та зарезервованих цілей інших роботів. Якщо шлях знайдено, робот резервує цільову позицію, що прилягає до палети (так звану «позицію підходу»), у системі координат, і виконує рух до неї. Він публікує це повідомлення в консолі, наприклад: «Робот #76: Прямую до позиції перед палетою 14 ((12, 5))»

Після досягнення палети робот симулює вилучення товару(викликає функцію pick\_item\_from\_pallet(взти товар з палети), яка приймає ідентифікатор палети, конкретний item\_id і кількість, яку слід забрати. Він виконує SELECT-запит, щоб з’ясувати поточний рівень доступного товару на цій палеті в таблиці inventory. Якщо товар є, виконується оновлення: якщо після зняття частини (між доступним на палеті й необхідним залишком) палета ще має залишок, тоді UPDATE inventory SET quantity = new\_qty; якщо ж зі зняттям усіх юнітів палета «спустошилась», то цей запис видаляється з таблиці.

У результаті цей метод повертає фактичну кількість одиниць товару, яка була переміщена до робота, у внутрішній список self.carrying\_items. Це означає, що якщо, скажімо, на палеті було 5 одиниць, а потрібно було 3, метод поверне 3, і remaining(залишок) відповідно зменшиться на 3 одиниці. В реальній фізичній системі робота це відповідало б акту дії маніпулятора або захоплювача. У симуляційній версії реалізація зводиться до оновлення кількості товару у кошику робота.

Робот продовжує процес збору товарів до тих пір, поки не буде виконано всі позиції поточного замовлення, або поки не буде вичерпано запас енергії, що передбачає перехід до зарядки (описано в розділі 4.6). У разі, якщо робот не може знайти палету з потрібним товаром, замовлення позначається як частково виконане або скасовується.

Таким чином, логіка взаємодії з інвентарем спроєктована таким чином, щоби забезпечити:

* 1. **гнучкість** у виборі палет з товаром (алгоритм завжди обирає найближчу можливу),
  2. **безпеку** через систему резервування підходу,
  3. **узгодженість даних** завдяки прямій синхронізації з базою після кожної операції,
  4. **масштабованість,** оскільки логіка не залежить від кількості роботів чи палет.

### 4.4.4 Взаємодія з полицями (резервування, доставка)

Щоб забезпечити коректну доставку товару на полиці з урахуванням резервування та уникнення конфліктів між кількома роботами, використовується спеціальний метод deliver\_items\_to\_shelf(доставка товару до полиці). У межах його виконання робот робить до п’яти спроб знайти відповідну полицю для розміщення товару. На кожному кроці викликається метод find\_and\_reserve\_free\_shelf(знайти і зарезервувати вільну полицю), який або повертає координати придатної полиці, або None, якщо знайти полицю не вдалося.

На початку метод перевіряє, чи вже є раніше зарезервованига полиця для цього замовлення зі статусом «busy». Якщо така полиця існує, вона відразу використовується. В іншому випадку відбувається пошук абсолютно вільних полиць, тобто таких, які не містять жодного товару та ще не були зайняті іншими роботами. Серед знайдених полиць обирається та, яка розташована найближче до поточної позиції робота — для цього обчислюється відстань до кожної доступної точки, і вибирається мінімальна. Після вибору робот намагається зарезервувати цю полицю в базі даних. Якщо інший робот встиг це зробити раніше, резервування вважається невдалим, і запускається нова спроба з невеликою випадковою затримкою.

Після отримання координат полиці робот викликає метод get\_approach\_position (отримати точку підходу до полиці) для визначення безпечної точки наближення. Цей метод враховує заздалегідь визначені підходи та, у разі блокування всіх маршрутів, чекає до 30 секунд. Якщо підходи залишаються зайнятими, робот шукає інші позиції в межах певного радіуса. Знайшовши доступне місце, робот рухається до нього та переходить до розміщення товару через метод place\_items\_on\_shelf(розміщення товару на полицю).

Усередині цього методу перевіряється, чи вже не знаходяться на вибраній полиці товари з іншого замовлення. Якщо так, і ідентифікатори замовлень не збігаються, система сигналізує про критичну помилку, і робот намагається скасувати зарезервовану дію. Якщо полиця вільна або містить товари цього ж замовлення, робот додає нові дані до бази інвентаря та видаляє переміщені товари зі свого внутрішнього списку. На завершення статус полиці оновлюється на «зайнята», що забороняє її подальше використання іншими роботами.

## 4.5 Навігація та уникнення колізій

Розробка алгоритму руху роботів у щільному середовищі має ключове значення для забезпечення безпеки та ефективності роботи багатоагентної системи на складі, де одночасно діють десятки пристроїв. У цьому розділі буде реалізовано модуль уникнення колізій, який здійснюватиме динамічне резервування клітинок простору, аналіз конфліктних ситуацій та автоматичне перенаправлення роботів у разі необхідності. Крім того,розглянемо інтегровані алгоритми пошуку.

### 4.5.1 Реалізовані алгоритми пошуку шляху та інтеграція у логіку руху

Як було закладено ще на етапі початкового проєктування, у системі реалізовано два класичні алгоритми пошуку шляху: A\* та алгоритм Дейкстри. Обидва алгоритми працюють по єдиному інтерфейсу, який інкапсулюється в компоненті Navigator. У додатку X наведено повний код їхньої реалізації. Кожен із алгоритмів на етапі планування шляху отримує такі вхідні параметри: координати початкової точки (x₀, y₀), координати цілі (xg, yg) і функцію check\_occupied(x, y), яка повертає True, якщо комірка зараз недоступна (через стелаж, палету або інший робот уже зарезервував чи запланував у неї хід).

Алгоритм A\* працює за принципом одного з найпоширеніших евристичних методів: для кожної сусідньої від поточної позиції клітинки він обчислює функцію

*f(n) = g(n) + h(n), де g(n*) (4.1)

де, вартість шляху від стартової вершини до n (визначається як кількість кроків або вагова сума), а h(n) — евристична оцінка відстані від n до цілі (в моїй системі використовується евклідова відстань). Такий підхід дозволяє знайти відносно короткий шлях дуже швидко. Дейкстра ж є «універсальнішим» у тому сенсі, що не потребує евристичної функції: він проглядає всі вершини на відстань g(n) у порядку зростання вартості, гарантовано повертаючи найдешевший шлях незалежно від топології графа. Однак на великих відстанях без евристики час потребує значно більше ресурсів, що потім я продемонструю у пункті «4.7. Моделювання та опис результатів»

Фактичні реалізації обох алгоритмів у Navigator об’єктно—орієнтовані: кожен метод зберігає статистику викликів — скільки разів алгоритм застосовувався, скільки згенеровано успішних шляхів, середня довжина та середній час виконання. Ці дані збираються й передаються у вигляді словника.

Вибір алгоритму відбувається автоматично на основі накопиченої статистики ефективності, або його може встановити адміністратор вручну через виклик методу set\_pathfinding\_algorithm(«a\_star») чи set\_pathfinding\_algorithm(«dijkstra»). Спочатку, якщо загальна кількість викликів кожного алгоритму менша за встановлений поріг auto\_switch\_threshold = 20, використовується значення за замовчуванням (self.pathfinding\_algorithm = «a\_star»). Після того, як обидва алгоритми були викликані принаймні 20 разів, проводиться порівняння їхніх оцінок якості: розраховується комбінований показник.(Рисунок 4.4)

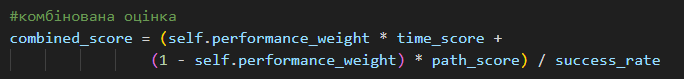


Рисунок 4.4 — Розрахунок оцінки для вибору алгоритма пошука

де performance\_weight(вага продуктивності) = 0.7 — це вага, з якою враховується час виконання проти довжини шляху, а success\_rate(оцінка успішності) = successful\_paths(успішні шляхи) / calls(кілкість викликів). Той алгоритм, в якого менший score, стає обраним для подальших маршрутів. Таким чином система сама навчається, який метод швидший або генерує коротші траєкторії відповідно до поточної завантаженості сітки й кількості зіткнень.

Зі свого боку, адміністратор може примусово зафіксувати вибір, через виклик robot.set\_pathfinding\_algorithm(«a\_start» або «dijkstra»).

Окрім власного “обчислення” маршруту, кожного разу перед викликом A\* або Dijkstra реалізовано підготовчу стадію: до моменту запуску планувальника у робота вже є актуальна «карта зайнятих клітин» через об’єднання інформації з reserved\_cells(зарезервовані\_клітини), robot\_destinations,( призначені\_цілі\_роботів) pallet\_coords(координати\_палет) і shelf\_coords(координати\_поличок). Саме ця інформація передається у вигляді callback-функції у check\_occupied(x, y). Тобто якщо якийсь робот щойно зарезервував клітинку, алгоритм A\* або Dijkstra вже не плануватиме через неї маршрут.

Після того, як обрано оптимальний алгоритм (A\* чи Dijkstra) і побудовано послідовність координат від поточної позиції до цілі, безпосередньо під час переходу до руху викликається універсальний метод move\_to\_basic(destination). Саме move\_to\_basic відповідає за поетапний рух вздовж обчисленого шляху, за кожен крок резервуючи наступну клітинку, перевіряючи зайнятість, оновлюючи позицію робота в базі даних та візуалізації (глобальні змінні reserved\_cells і robot\_destinations), і, у разі необхідності, запускаючи обробку помилок (взаємоблокувань чи повторного перерахунку маршруту). Таким чином, move\_to\_basic є єдиною точкою входу з навігатора у реальне виконання маршруту в середовищі складу.

У випадках, коли рух по обчисленому шляху не вдається (наприклад, через образне зайняття цільової клітинки іншим роботом, що ще не звільнив її), викликається «друга стадія» навігації, а саме окремий алгоритм \_try\_alternative\_route. Він використовує внутрішню модифікацію A\* із «штрафами» за кожен сусідній зайнятий квадрат (5 одиниць штрафу за кожне послідовне сусідство), що дозволяє генерувати шлях, який не пролягає у густо заблокованих областях сітки. Якщо ж і альтернативного рішення не знайдено, робот через випадкову паузу повертається до повторного виклику move\_to\_basic(destination), після чого він або успішно зарезервує новий шлях, або згодом увімкне механізм дедлока.

Таким чином, реалізація алгоритмів пошуку шляху поєднує класичний набір A\* і Dijkstra з динамічним вибором методу на основі накопичених статистичних даних або налаштувань адміністратора, а також із додатковою обробкою «штрафів» за потенційну зайнятість сусідніх клітинок. У сукупності це дає гнучку й адаптивну навігацію, яка реагує на змінну обстановку всередині складу та мінімізує час простоїв при одночасній роботі великої кількості роботів.

### 4.5.2 Детекція і розв’язання deadlock—ситуацій

### 

У багатокористувацькому середовищі, де кілька автономних роботів одночасно рухаються й виконують завдання в одній сітці, нерідко виникають ситуації, коли двоє або більше машин опиняються ніби “замкненими” між собою: кожна з них чекає, що звільниться комірка, яку має зайняти інший, а той, у свою чергу, чекає звільнення іншої клітинки від першого. Саме так трапляється класичний deadlock (мертвий стан), коли без узгодженої стратегії жоден із роботів не може продовжити рух. У нашій системі для перехоплення й розв’язання таких тупикових ситуацій реалізовано низку взаємопов’язаних механізмів: по—перше, виявлення циклічних залежностей (аналіз “ланцюга” очікування), по—друге, встановлення пріоритетів між учасниками конфлікту та стратегія відходу, а по—третє, пошук альтернативних маршрутів, коли стандартний шлях більше не забезпечує безпечного проходу. Нижче наведено докладний опис цих кроків, із посиланням на відповідні фрагменти коду в Додатку Х.

Перш ніж говорити про те, як робот “розпізнає” мертвий стан, нагадаємо, що в кожному кроці руху, перед тим як перейти до нової клітинки next\_pos, він перевіряє через is\_cell\_occupied(next\_pos) (яка враховує інформацію з глобальних структур reserved\_cells та robot\_destinations) – чи не зайнята ця клітинка іншим агентом або чи випадково хтось не запланував сюди хід. Якщо метод повернув True, тобто поле next\_pos виявилося зайнятим, робот не одразу скасовує весь маршрут, а спершу починає очікувати на тій самій позиції кілька разів із незначними затримками (зазвичай 0.8 секунди на ітерацію). Через внутрішню логіку підрахунку часу очікування, в який робітник захований у змінній consecutive\_wait\_time, якщо цей час під час очікування на одній і тій самій цілі перевищив критичний поріг (2 секунди), робот припиняє чекати сам по собі й генерує запит на детекцію можливого deadlock’а. Сам виклик виглядає так(Рисунок 4.5):

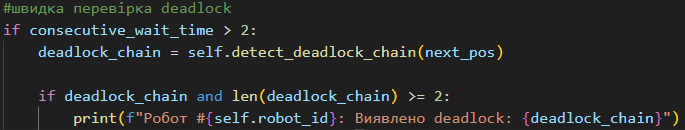


Рисунок 4.5 — умова виклику перевірки на мертвий стан

#### 4.5.2.1 Аналіз залежностей

Реалізація методу detect\_deadlock\_chain(self, target\_pos)(див. Додаток А) ілюструє, як саме виявляється “циклічна” залежність між роботами. Роботу, який викликає цю функцію, ми назвемо поточним (self.robot\_id), а target\_pos — координатами тієї клітинки, до якої він намагався перейти, але вона виявилася зайнятою. На першому кроці в локальній структурі даних ми фіксуємо, що поточний робот чекає на target\_pos. Далі, у межах блоку with grid\_lock, метод спочатку визначає, який робот саме захопив цю клітинку (він звертається до внутрішньої структури, де зберігаються всі наразі зарезервовані позиції). Якщо виявляється, що ця клітинка зайнята іншим роботом (назвемо його «робот, який блокує»), і ця машина відрізняється від поточного робота, її ідентифікатор записується у список ланцюжка блокувань. Далі вважається, що цей «робот—блокатор» може мати власну цільову клітинку, яку він планує зайняти наступною. Тому перевіряється, чи існує інформація про те, куди саме він збирається рухатися. Якщо така інформація є, дістається координата його наступної цілі. Якщо виявляється, що ця клітинка вже входить до переліку тих, які ми раніше відвідали під час пошуку циклу — це означає, що ми повернулися в ту саму точку, і таким чином з’явився замкнений ланцюг зависань. У такому випадку метод повертає список, який починається з ідентифікатора поточного робота, далі містить ідентифікатори всіх роботів, що стоять у цьому колі блокувань «[self.robot\_id, blocking\_robot, …]», який означає: “робот №А чекає, поки робот №Б звільнить позицію, а Б, у свою чергу, чекає, поки С звільнить свою позицію, …, а останній чекає на А” словом, замкнене коло. Якщо десь на проміжному кроці виявляється, що блокуючий робот не має власної цілі в robot\_destinations (тобто він не планував рух далі, просто займав комірку), то функція одразу повертає None, бо мертвого стану не утворюється, треба було просто зачекати, поки хтось звільнить ціль.

Таким чином, кожного разу, коли поточний робот стоїть у зоні очікування понад 2 секунди, відбувається рекурсивна перевірка: “а якщо мені не пощастило, робот, який мене блокує, також блокує інший, а той інший блокує ще когось?” — і якщо в результаті знайшлися два чи більше учасники, що образили “мертвий ланцюг”, цей ланцюг передається далі, куди його вже обробляє логіка розв’язання. Відповідна блок схема, яка візуалізує це процес представлена на кресленику ІК12.310БАК.006 Д6 .

#### 4.5.2.2 Опис вибору пріоритетів між роботами

Коли функція детекту виявляє, що deadlock\_chain = [id₁, id₂, …, idₖ] (де перший елемент — це id робота), наступний крок — вирішити, хто ж із учасників «повинен перший відступити». Для випадку, коли довжина ланцюга(len(deadlock\_chain)) = 2 (тобто тільки двоє роботів, A й B, стали блокуючими одне одного), у коді впроваджено просте, але ефективне правило: перший відходить робот із більшим Id. Наприклад, якщо дедлок виник між роботом 82 і роботом 77, то, оскільки 82 > 77, саме #82 бере на себе відповідальність зробити відхід. Це правило детерміноване та однозначно знімає “гонку” двох симетричних агентів, які інакше могли б розмірковувати «а може, нехай першим піде той» і виявитися в стані «livelock» (або очікувати нескінченно довго).

Якщо ж у робота з більшим Id нічого не виходить, тобто йому не вдалося знайти вільної клітини, щоб відступити, то він одразу фіксує себе в глобальному множині robots\_cannot\_retreat. Це означає приблизно таке: «Я спробував відхід, але навколо мене немає жодного вільного місця, тому зараз я знімаюся з гри, і пріоритет на відхід тепер має інший робот». Усі інші учасники deadlock (тут той, що з меншим Id) перевіряють, чи міститься їхній суперник у robots\_cannot\_retreat; якщо так, то маленький Id автоматично отримує право відступити, навіть якщо за звичайних обставин він би чекав спочатку, щоб дати дорогу більшому Id. У випадку, якщо і другий учасник дедлока не може відступити, виникає “критична” ситуація: обидва позначені як “не можуть відступити”. Тоді обидва машини виконують коротку паузу (time.sleep(5)), після чого кожен із них намагається ще раз викликати move\_to\_basic(destination) (тобто перерахувати свій маршрут заново, намагаючись знайти обхід). Це дає шанс виявити нове вікно, коли хтось із інших роботів уже змінив своє призначення, або з’явилися нові вільні клітинки, і дозволяє уникнути “кам’яного” deadlock.

У випадку, коли у виявленому “ланцюзі” більше ніж два учасники (наприклад, A → B → C → A), весь ланцюг передається вниз. Кожен з роботів знаходить себе всередині цього списку й обчислює свій власний індекс my\_index = deadlock\_chain.index(self.robot\_id). Далі застосовується правило: якщо my\_index % 2 == 0 (тобто його позиція в циклі парна), або якщо кількість спроб розв’язати deadlock (deadlock\_resolution\_attempts) вже перевищила 2, то саме цей робот має першим зробити ретрит. Іншими словами, намагаючись розірвати «кільце” із трьох і більш машин, парні індекси й учасники із вичерпаними лімітами спроб вважаються підготовленими до відходу. Решта роботів у цьому циклі «зупиняються й чекають», у два рази збільшуючи змінну deadlock\_resolution\_attempts та даючи час «парним» здійснити маневр відступу. Якщо ж жоден із «парних» не може відступити (бо теж не було вільних клітинок), вони просто вказують «не вийшло», і автоматично через кілька ітерацій (через підвищений deadlock\_resolution\_attempts) право відступу перейде до інших учасників (включно з непарними). Така схема з чергуванням ролей «парний—відступає» дає змогу розірвати цикличну залежність, коли одночасно залучено більше ніж двоє роботів.

#### 4.5.2.3 Стратегія відходу

Що робить робот, коли він отримав вирок «я повинен відступити?» По—перше, в методі \_execute\_retreat\_maneuver(self, original\_destination) відбувається така послідовність:

1. Визначення своєї поточної позиції залишається в self.current\_position.
2. Звільнення поточної клітинки шляхом виклику self.release\_cell(current\_x, current\_y) та одночасним видаленням з robot\_destinations, якщо запис про цю машину існував. Важливо: саме цей крок миттєво звільняє простір для іншого робота, який очікував зайняти цю ж комірку.
3. Пошук вільних клітинок навколо, виклик методу find\_nearest\_free\_cells(count=5), який обходить навколо self.current\_position з радіусом 1, 2, 3, … доки не знайде принаймні 5 вільних позицій. У результаті повертається список із, скажімо, 3–5 клітинок—кандидатів, відсортованих за відстанню до початкової позиції.
4. Вибір випадкового і безпечного місця — береться перші три з цього списку й рандомно обирається одна позиція, щоб уникнути конфлікту, коли декілька машин одночасно відступають. Якщо список зовсім порожній, виводиться повідомлення “Немає вільних клітинок для відступу — залишаюся на місці”, пауза приблизно 2 секунди, і метод повертає False, бо неможливо здійснити ретрит.
5. Резервування нової позиції (if self.reserve\_cell(\*retreat\_pos): …), якщо вдалося забронювати ту комірку, робот переходить у неї, оновлює базу даних через self.update\_position(\*retreat\_pos) і зменшує рівень батареї. Далі йде пауза 2–4 сек (функція time.sleep(random.uniform(2, 4))), щоб дати іншим учасникам дедлока достатньо часу пройти свої кроки.
6. Звільнення тимчасової позиції, після паузи робот робить self.release\_cell(\*retreat\_pos) й ще одну коротку затримку (0.5–1.5 сек), щоб переконатися, що інші учасники дедлока точно рухнули далі.
7. Повернення до початкового завдання, цей крок означає повторний виклик self.move\_to\_basic(original\_destination), тобто спробу виконати рух до тієї самої цілі, з якої розпочався deadlock. Якщо й цього разу виникнуть нові блокування або колізії, алгоритм дедлока знову відпрацює, але вже з оновленою інформацією про зайняті клітинки та, можливо, з іншим ланцюжком залежностей. Увесь код демонструється у додатку(Додаток А).

Якщо ж на етапі резервування позиції для відходу reserve\_cell(\*retreat\_pos) поверне False (тому що за мілісекунду до цього інший робот встигне заблокувати цю клітинку)метод \_execute\_retreat\_maneuver уже розуміє, що «я зараз не зможу ні відступити, ні зайняти жодну сусідню клітинку». У цьому разі робот записує себе в глобальну змінну robots\_cannot\_retreat (якщо ще там не був) і повертає False, це сигналізує тій логіці, що вирішує, що його «пробували змусити відступити, але не вийшло» і що тепер потрібно дати право відпочити іншому учаснику дедлока (для випадку 2 учасників) або залишити його в стороні, доки ситуація не зміниться.

Отже, підсумок стратегії розв’язання таких ситуацій такий:

1. Спочатку детектується ланцюжок залежностей через detect\_deadlock\_chain.
2. Якщо виявлено двох роботів, пріоритет на відхід отримує той, у кого більший Id, але якщо в нього не вийшло, це позначається у robots\_cannot\_retreat і тоді уступає місце менший Id.
3. Якщо виявлено трьох і більше, пріоритетним учасником є той, чий індекс у списку deadlock’а парний, або той, хто вже тричі спробував вирішувати ситуацію (в такому разі роботи вже не ділять на парних/непарних).
4. Якщо ні у кого не вийшло відступити (бо всі оточені іншими роботами або преградами), усі учасники дедлока (двоє чи більше) через заданий інтервал (time.sleep(5)) повторно запускають move\_to\_basic(destination), тобто пробують перерахувати маршрут в оновленій ситуації.

Якщо підсумувати, весь механізм дедлока будується навколо ідеї **«перевірка довгого очікування → аналіз ланцюга блокувань → розподіл пріоритету на відхід → маневр ретриту → повторний відлік»**. Це дозволяє уникнути «кам’яних» ситуацій, коли роботи могли б стояти вічно, чекаючи один на одного. Завдяки тому, що пріоритет визначається детерміновано (по Id чи позиції в циклі) і що кожен робот у разі невдалого відходу сигналізує про свою не можливість відійти(robots\_cannot\_retreat), система гарантує, що **хоча б один з учасників дедлока завжди знайде собі вільне місце й виконає маневр,** руйнуючи тим самим «мертве коло».

Усі згадані фрагменти коду з реалізацією вивлення ланцюга залежносты, логікою вибору пріоритетів і викликом відходу наведені в Додатку Х.

## 4.6 Система керування батареєю і зарядкою

У системі автономних роботів, призначених для роботи на складі чи в логістичному середовищі, управління енергетичними ресурсами відіграє критично важливу роль. На відміну від безперервно під’єднаних до джерела енергії станків чи конвеєрів, кожен робот покладається на власний акумулятор, ємність якого обмежена. Якщо механізм контролю заряду не буде реалізовано на достатньо високому рівні, робот може опинитися посеред маршруту без можливості повернутися до зарядної станції, що призведе до зупинки всієї системи обробки замовлень і простою обладнання.

Перш ніж перейти до деталей, зазначимо ключові задачі системи керування батареєю:

* **Моделювання реального споживання енергії** під час руху;
* **Прийняття рішення про необхідність зарядки** ще до початку виконання будь—якого завдання, з урахуванням довжини маршруту та поточного заряду;
* **Вчасний перехід у критичний режим**, коли рівень заряду падає до загрозливо низьких значень;
* **Контроль фізичних властивостей процесу зарядки**, включно з динамікою потужності та ефективністю;
* **Автоматичне повернення до роботи** після того, як акумулятор відновився вище безпечного рівня.

### **4.6.1. Моделювання реального споживання енергії** під час руху

У нашій реалізації кожен крок робота по сітці розглядається як проїзд на одну клітинку, що еквівалентно руху на 1 метр. Щоб наблизити модель витрат енергії до реальних фізичних процесів, використано наступну формулу:

1) Сумарна маса робота й вантажу:

(4.2)

2) Сила опору тертям (горизонтальна складова):

(4.3)

3) Механічна енергія, витрачена колесами (Вт\год) при русі на d метрів

(4.4)

4) З урахуванням ефективності трансмісії:

(4.5)

5) Переведення в проценти від номінальної ємності акумулятора CWh:

(4.6)

6) Додаткові умовні витрати(електроніка, прискорення, гальмування):

(4.7)

7) Сумарний відсоток витрат на один крок (1 метр):

*L* (4.8)

Пояснення параметрів:

* — це власна маса порожнього робота(наприклад 50кг);
* — це середня маса вантажу(наприклад 20кг) ;
* — це безрозмірний коефіцієнт тріння об покриття , який дорівнє 0.02;
* *g* — це прискорення вільного падіння () ;
* *d* — це відстань, яка дорівнює 1 метру;
* — це загальний ККД трансмісії , який дорівнює 0.9;
* — це номінальна ємність акумулятора в Вт·год, яка дорівнює ;
* — це умовний «процент втрат» на метр, що моделює витрати на електроніку, прискорення/гальмування (який дорівнє 0,01  % на метр).

Програмна реалізація надана у додатку (Додаток А)

Формула для оцінки витрати заряду максимально наближена до реальних фізичних процесів з кількох причин. По—перше, вона починається з урахування сумарної маси робота разом з вантажем, що відображає реальний вага­вий опір при русі по підлозі. Далі в ній враховується горизонтальна сила тріння, яка обчислюється як добуток коефіцієнта тріння й маси (помноженої на прискорення вільного падіння), тобто це імітує ту фізичну силу, з якою колеса долають контакт із поверхнею підлоги. Ця сила помножується на пройдений шлях, що дає величину механічної енергії (в ват—годинах), яку витратили саме колісні приводи. Наступним кроком до отриманої механічної енергії застосовується поправка на втрати в передаточному механізмі, тобто враховується, що не вся енергія з батареї дійде безпосередньо до коліс: частина «загубиться» на ККД трансмісії. Після цього результат конвертується у відсоток від загальної ємності акумулятора, що дозволяє оцінити, яку частину заряду реально спожито під час руху. І нарешті до цього базового відсотка додаються умовні додаткові втрати на електроніку, прискорення й гальмування — вони моделюються як сталий відсоток за кожен метр пробігу.

У підсумку відсоток витрат на один метр руху розраховується як поєднання трьох складових:

* Підрахунок механічної енергії колісного приводу, виходячи з реальної ваги робота з вантажем та сили тріння, приведеної до ват—годин і скоригованої з урахуванням КПД трансмісії;
* Переклад цієї механічної енергії у відсоток від номінальної ємності батареї, що відображає, яку частину акумулятора реально витрачено;
* Додавання фіксованого відсотка на кожен метр для покриття всіх додаткових витрат (електроніка, динаміка прискорень і гальмувань);

Така модель максимально відповідає фізичним умовам руху: вона показує, що чим важчий робот і вантаж, чим більший коефіцієнт тертя або менший у передачі ККД, тим більше ват—годин буде витрачено. Переведення у відсотки робить цю інформацію безпосередньо зрозумілою для системи керування: можна точно прогнозувати, чи вистачить заряду для заданого маршруту, і своєчасно вирішувати, чи варто їхати виконувати замовлення або спершу повертатися до зарядної станції.

### 4.6.2. Прийняття рішення про необхідність зарядки

Перед тим як брати будь—яке замовлення, система спершу оцінює, чи вистачить поточного заряду для повного виконання маршруту «поточна позиція → палета → полиця → стандартна позиція» з урахуванням ваги вантажу та додаткових втрат. Алгоритм складається з таких кроків:

1. Побудова маршруту до палети без вантажу. Викликається функція пошуку шляху від поточної позиції робота до координат палети, отриманих із БД. Якщо маршруту немає (наприклад, через колізії), система одразу відмовляється від взяття цього замовлення. Інакше обчислюється довжина шляху (кількість клітинок, які відповідають метрам), помножена на енерговитрату за метр без вантажу. Під час цього враховуються: маса порожнього робота, коефіцієнт тертя, ефективність трансмісії, номінальна ємність акумулятора та умовні додаткові втрати на електроніку й уповільнення—розгон. Результатом є відсоток заряду, необхідний, щоб дістатися до палети.
2. Побудова маршруту від палети до полиці з вантажем. Після прибуття до палети вказаний вантаж (середньої ваги, наприклад 20 кг) забирається, і система викликає метод знаходження підходу до полиці (з урахуванням колізій і зарезервованих клітин) та побудову шляху до неї. Довжина цього шляху також переводиться в метри, і за допомогою тієї самої фізичної формули (але з доданою масою вантажу) обчислюється відсоток заряду, який споживається під час під’їзду до полиці з вантажем.
3. Повернення від полиці до стандартної позиції. Після доставки вантажу робот вважається звільненим від ваги (або ж повернення може відбуватися з мінімальним залишком), тому останній відрізок від полиці до стандартного місця очікування вираховується з масою без вантажу. Знову знаходиться шлях, визначається довжина та обчислюється відсоток енергії, потрібний на цей етап.
4. Порівняння сукупних витрат із доступним зарядом та запасом. Сума всіх трьох відсотків (до палети + до полиці + назад) порівнюється з поточним рівнем заряду, за мінусом встановленого безпечного запасу (наприклад, 15 %). Якщо після виконання всіх трьох етапів заряду в робота залишиться менше за цей поріг, метод повертає «False», і робот не бере це замовлення, а одразу прямує на зарядку. Інакше – метод повертає «True» і дозволяє роботам почати рух до палети. У коді ця логіка реалізована у функції can\_complete\_order\_with\_battery(...), до якої передаються координати палети та приблизна вага вантажу.
5. Додаткова умова перевірки перед стартом. Якщо в базі даних немає жодного замовлення або всі доступні замовлення провалюють перевірку енерговитрат, у головному циклі (run()) робот звертається на зарядну станцію, лише коли поточний заряд падає нижче верхнього порогу («якщо немає замовлень і заряд < 90 % → їхати заряджатися»). Таким чином, навіть якщо нові замовлення з’являться під час повернення на стандартну позицію, система повторно виконає перевірку батареї перед тим, як рушити до палети, і, за необхідності, переорієнтується на зарядку.

Завдяки цьому підходу робот гарантує, що бере лише ті завдання, які він фізично зможе завершити без ризику розрядитися посеред шляху. На практиці такий алгоритм дозволяє уникнути зупинок у межах робочої зони та забезпечує безперервність обробки замовлень у складській логістиці.(Відповідний реалізований код надан у додатку (Додаток А))

### 4.6.4. Процес зарядки: фізична модель та алгоритмічна реалізація

Після того як робот приняв рішення прямувати на зарядку, наступним етапом було реалізувати реалістичний(фізичний) процес зарядки батареї робота. Так ось, коли робот переходить у режим зарядки, він припиняє витрачати енергію на рух і натомість починає приймати її від зарядної станції. У реальних умовах потужність зарядного пристрою обмежена: кожна секунда заряджання поповнює акумулятор на певний відсоток, що залежить від номінальної потужності зарядного пристрою, ККД системи заряджання та загальної ємності батареї.

Після прибуття до зарядної станції (процес докладно описано в коді функцією, яка переміщує робота до координат станції та переводить його в стан «charging» у Додатку А) починається безперервний цикл зарядки. Фізично це означає, що зарядний пристрій подає електричну потужність у батарею: частина цієї енергії безпосередньо накопичується в хімічному заряді акумулятора, а частина втрачається через нагрівання й інші ефекти.

Для того, щоб змоделювати фізичний процес заряджання, та оцінити швидкість відновлення заряду батареї за 1 секунду, було використано відповідну формулу:

(4.9)

* де потужність зарядного пристрою у Вт(що у реалізації дорівнює 1000 Вт на год., що відповідає потребам швидкої зарядки)
* це — ККД зарядної системи
* це — ємність акумулятора в Вт—год

У результаті отримуємо кількість відсотків на які зарядиться акумулятор за 1 секунду. Таким чином процес зарядки моделюється як поступове лінійне зростання рівня заряду батареї. У процесі підзарядки зчитується кількість ват—годин, які надходять від зарядного пристрою, і конвертуються в відсотки батареї, а кожне оновлення (наприклад, «Зарядка… 45.3 %») виводиться у консоль

Після завершення зарядки робот автоматично виходить із стану «charging» та повертається до нормальної роботи: як тільки рівень заряду досягає встановленого порогу, або якщо з’явилося замовлення, і по розрахункам робот може його виконати, тоді система переводить його в режим «idle», і він виходить з місця зарядки.

Підсумуємо: Під час руху та виконання завдань кожен крок конвертується в реальні відсотки витрат батареї з урахуванням ваги, коефіцієнта тріння, ефективності трансмісії, витрати на електроніку, прискорення/гальмування. Як тільки рівень заряду опускається до критичної позначки (15 %), робот негайно вирушає на попередньо зарезервовану зарядну станцію. Як тільки батарея відновлюється до встановленого рівня, робот завершує цикл зарядки, або якщо з’явилося замовлення, і по розрахункам робот може його виконати, надсилає повідомлення про відновлений стан («Готовий до роботи») та без зволікання повертається до робочого процесу.

### 4.6.5 Демонстрація реалізованой механіки зарядки

Для того, щоб продемонструвати реалізовані механіки зарядки, ініціалізуємо одного робота. Задамо йому поточний рівень заряду у – 10% (Рисунок 4.6) , створимо 1 замовлення(Рисунок 4.7) та запустимо симуляцію:

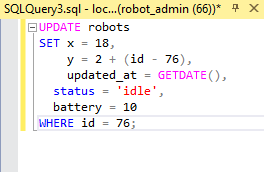


Рисунок 4.6 — Задаємо роботу 10% батареї

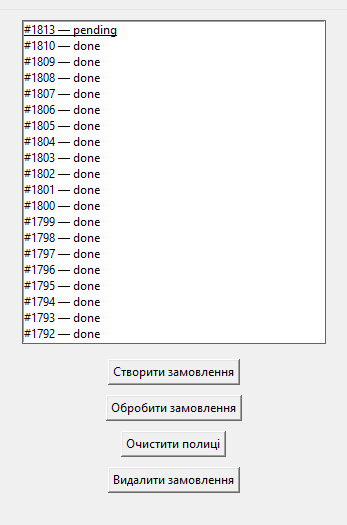


Рисунок 4.7 — Створюємо замовлення

Встановлено наступні параметри батареї та потужність зарядного пристрою:

* self.charging\_power\_W = 1000.0 – потужність зарядного пристрою, Вт
* self.battery\_capacity\_Wh = 1500.0 – номінальна ємність акумулятора робота, Вт.год

Запускаємо симуляцію та передивляємось логи у консолі(Рисунок 4.8):

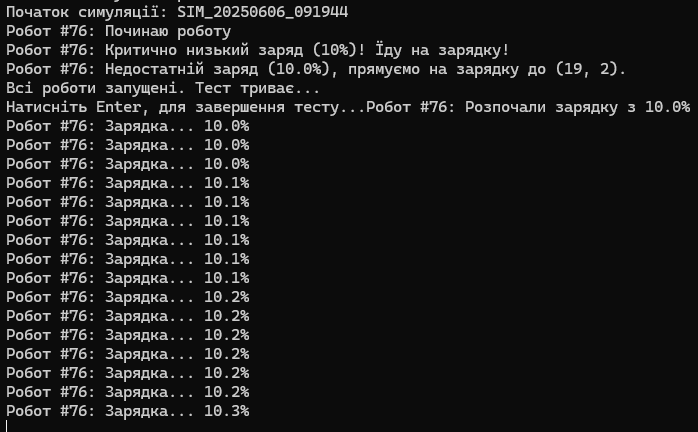


Рисунок 4.8 — Консоль з повідомленнями про успішний перехід на зарядку

Як ми бачимо, робот успішно зчитав свій поточний заряд та не став обробляти замовлення, а перейшов до станції зарядки, та розпочав заряджання акумулятору. Після приблизно 20 хвилин зарядки, робот відновив свій заряд до 30%, який встановлен у логіці системи як мінімальний рівень заряду, при якому робот може взяти до обробки замовлення, про що він звітує у консоль:

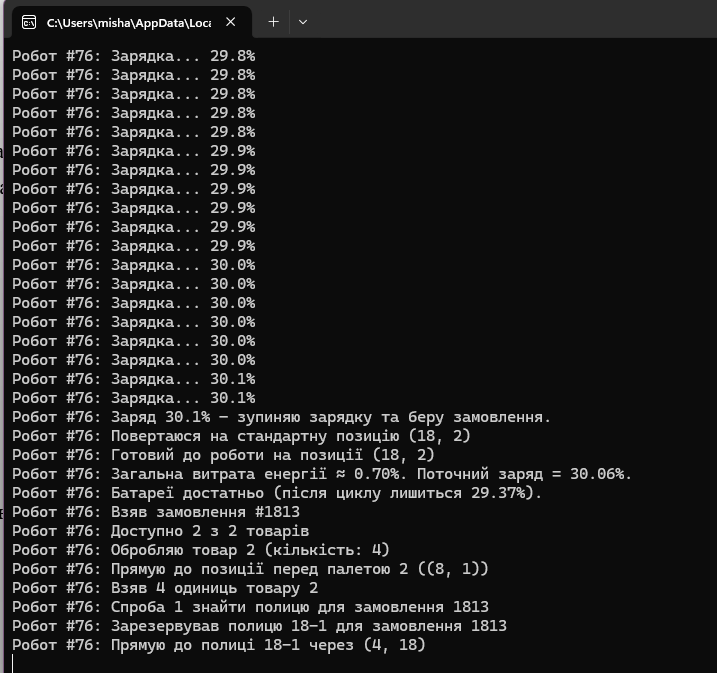


Рисунок 4.9 — Звіт у консоль про початок обробки замовлення

Як ми бачимо на рисунку 4.9 – робот успішно зчитав свій оновлений стан акумулятора та перейшов до виконання замовлення. На основі проаналізованого логу(Рисунок 4.10) роботи видно, що робот №76 успішно завершив обробку двох окремих замовлень – №1813 та №1814. Кожне з них було повністю виконане, що підтверджується повідомленнями про успішне збирання товарів, правильне розміщення їх на відповідні полиці, а також фінальним оновленням статусу замовлення. Важливо зазначити, що після завершення останнього завдання, робот виконав контроль перевірки рівня заряду батареї. Згідно з умовами безпечної експлуатації, якщо поточний рівень заряду менший за 100%, та не має замовлень для обробки – робот має пріоритетно перейти до фази зарядки.

У наведеному прикладі, при заряді 28.57 % робот автоматично ініціював перехід до призначеної зарядної станції, чим продемонстрував коректну реакцію на зміну внутрішнього ресурсу батареї.

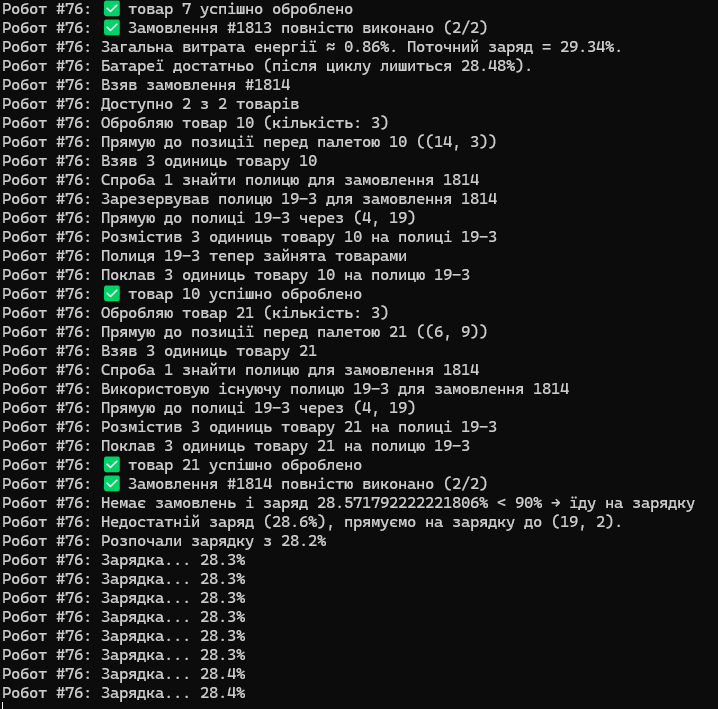


Рисунок 4.10 — Звіт про успішне виконання замовлення та перехід до зарядної станції

Можно зробити висновок що механіка та логіка переходу до зарядки успішно працює, та моделює реалістичний фізичний процес: заряд збільшується поступово, що наближає симуляцію до поведінки реальних роботизованих систем. Таке поєднання логічної точності та фізичної достовірності підсилює ефективність симуляційного середовища та підтверджує, що система зарядки спроєктована з урахуванням вимог до автономної робототехніки в реальних умовах експлуатації.

## 4.7 Моделювання та опис результатів

Після детального опису реалізованих механік і логіки функціонування системи, на цьому етапі доцільно перейти до моделювання, тобто перевірки працездатності всієї підсистеми керування роботами в умовах, наближених до реального середовища. Метою є імітація повноцінної роботи автоматизованого складу онлайн—магазину з багатьма активними замовленнями, паралельним виконанням завдань і обробкою різних логістичних сценаріїв.

У межах симуляції запускається 10 незалежних роботів(роботи 76—85), кожен із яких стартує зі 100 % заряду батареї. Послідовно буде згенеровано 50 тестових замовлень, які надходитимуть у систему для обробки в режимі реального часу. У процесі моделювання перевіряється низка ключових аспектів:

* наскільки ефективно розподіляються й виконуються замовлення,
* чи виникають колізії між роботами,
* як система поводиться у випадках нестачі товару на палетах,
* чи коректно працює логіка резервування полиць,
* чи функціонує механізм енергоспоживання відповідно до заданих умов.

Окрему увагу буде приділено аналізу ситуацій, у яких може виникнути критична зупинка роботів (deadlock) або збої в логістиці. В кінці експерименту буде представлено узагальнений звіт симуляції: час виконання замовлень, загальні статистичні метрики ефективності та висновки щодо надійності й адаптивності реалізованої системи.(Рисунок 4.13)

Для початку створимо 10 замовлень(рисунок 4.11) і запустимо симуляцію(рисунок 4.12), потім поступово будемо додавати, поки не дійдемо до 50 замовлень:

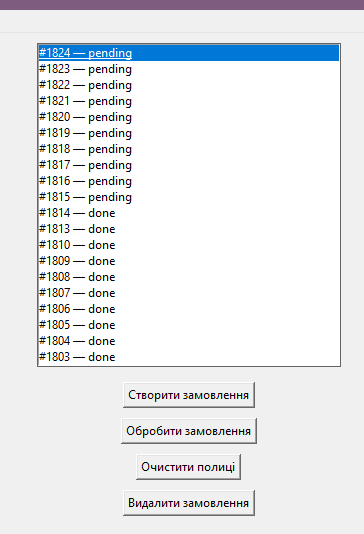


Рисунок 4.11 — Створення 10 замовлень



Рисунок 4.12 — Процесс обробки замовлень

У результаті аналізуючи логи у консолі, та створений звіт симуляції, 49 замовлень було успішно виконано та доставлено до полиць, з правильним резервуванням та оновленням бази даних, 1 замовлення не було оброблено, тому що товару не має у наявності, про що я отримав звіт у консоль. Також, роботи успішно подолали усі deadlock ситуації, наприклад:

* Робот #78: Виявлено deadlock: [78, 81, 83]
* Робот #78: Відступаю для вирішення складного deadlock;
* Робот #78: Виконую маневр відступу;
* Робот #78: Відступаю на (12.0, 9.0);
* Робот #78: Чекаю 3.2 секунд;
* Робот #78: Повертаюся до завдання»

Під час обробки замовлень система здебільшого застосовувала алгоритм **A\*(385 викликів, а середній час виконання = 2.47мс ),** який продемонстрував значно вищу швидкість побудови маршрутів порівняно з Dijkstra(17 викликів, **а середній час виконання = 9.92 мс**). Це підтверджується результатами симуляції, наведеними у звіті (рис. 4.13).

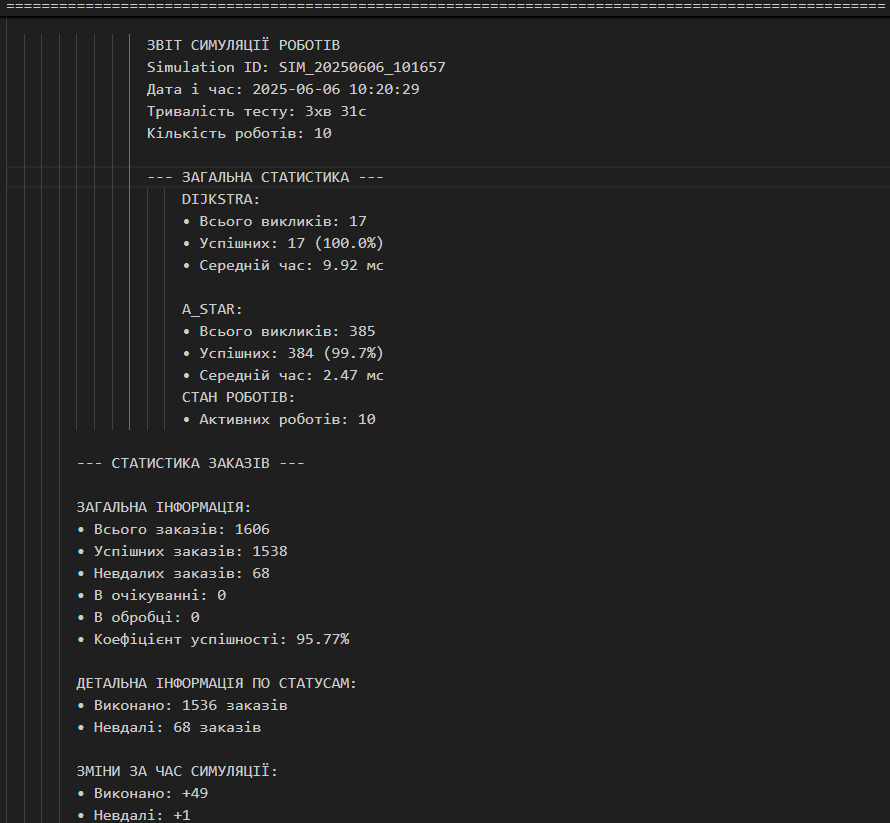


Рисунок 4.13 — Звіт статистики

Варто зауважити, що **швидкість переміщення роботів була навмисно збільшена** з метою скорочення часу симуляції. Це технічне рішення не впливає на достовірність або об’єктивність результатів моделювання. Крім того, **механізм контролю енергоспоживання** функціонував коректно: перед взяттям кожного нового завдання роботи перевіряли актуальний рівень заряду батареї та точно обчислювали необхідну енергію для виконання маршруту. На основі проведених тестувань можна зробити висновок, що використання алгоритму **A\*** є більш доцільним у межах розробленої системи. Завдяки використанню **евклідової евристики**, A\* забезпечує значно швидше знаходження оптимального маршруту в умовах динамічного середовища, не поступаючись при цьому точністю чи надійністю. Це особливо важливо для оперативної обробки великої кількості замовлень у режимі реального часу.

## 4.8 Висновок до розділу 4

У четвертому розділі було здійснено безпосередню реалізацію програмної системи управління мобільними роботами для складу. Розроблена архітектура ґрунтується на модульному підході, що забезпечує чітке розділення відповідальності між основними компонентами: логікою навігації, обробкою замовлень, управлінням енергоспоживанням, інвентаризацією, збором статистики та взаємодією з базою даних. Усі ці модулі були імплементовані з урахуванням вимог до масштабованості, стабільності й ефективності роботи в умовах реального часу.

У межах реалізації інтегровано два алгоритми пошуку шляху: A\* та Dijkstra, що дозволяє системі динамічно обирати оптимальний маршрут залежно від поточної ситуації на складі. Було впроваджено механізми розпізнавання та вирішення конфліктів, таких як колізії та deadlock-сценарії, а також логіку енергоменеджменту, що дає змогу роботам самостійно оцінювати свій заряд і приймати рішення про повернення до зарядної станції.

Окрему увагу приділено візуалізації логіки поведінки роботів: для цього розроблено блок-схему станів робота, що представлена на кресленику ІК12.310БАК.006 Д7. Вона відображає основні стани, в яких може перебувати робот (очікування, виконання замовлення, заряджання, переміщення тощо), та переходи між ними залежно від умов середовища і внутрішнього стану системи. Ця схема допоможе у розумінні послідовності дій та принципів роботи кожного автономного робота в системі.

У результаті реалізовано гнучку, розширювану та логічно узгоджену систему, яка забезпечує ефективне функціонування роботів у складському середовищі з високим рівнем автоматизації.

# **РОЗДІЛ 5.**

# **ВИСНОВКИ**

Підсумовуючи результати виконаної роботи, слід зазначити, що основна мета проєктування підсистеми автоматизованого керування рухом мобільних роботів на продовльчому складі онлайн магазину, полягала в створенні цілісного програмного комплексу. Здатного забезпечити автономну навігацію до десяти роботів, ефективний розподіл і виконання замовлень, контроль інвентарю, а також збір і генерацію статистики роботи всіх компонентів. Виконані дослідження, моделювання та реалізація підтвердили можливість досягнення поставлених цілей: розроблено такі складові, як база даних для збереження інформації про замовлення й інвентар, алгоритми формування маршрутів із врахуванням запобігання колізіям, логіку обробки замовлень із паралельним розподілом між роботами, модуль управління енергоспоживанням, а також механізми збору й аналізу статистичних даних.

Щодо реалізації кожної з очікуваних функціональних вимог (розділ 2.3.1), можна констатувати наступне. По-перше, система підтримує одночасну роботу до десяти роботів: у середовищі симуляції створено десять віртуальних агентів, кожному з яких присвоєно унікальний ідентифікатор та прив’язано відповідну зарядну станцію. Роботи автономно пересуваються за динамічно згенерованими маршрутами у реальному часі, відстежуються їхні наявні позиції через візуалізацію сітки складу. Таким чином, вимоги з групи «Управління роботами» було виконано в повному обсязі. По-друге, у частині «Обробка замовлень» реалізовано автоматичну генерацію тестових замовлень для симуляції, їхній розподіл між вільними роботами, паралельну обробку кількох замовлень одночасно, а також механізм «збирання» товарів із палет і доставки їх до полиць. Статуси замовлень («pending», «processing», «done», «failed», «partial») фіксуються у базі даних. По-третє, пункт «навігація та планування маршрутів» передбачав можливість автоматичного вибору оптимального алгоритму (наприклад, алгоритму A\* або Dijkstra), запобігання колізіям за допомогою динамічного резервування клітинок, виявлення й розв’язання deadlock-ситуацій та пошук альтернативних маршрутів у разі блокування шляху, що було успішно реалізовано. А також реалізовано базову процедуру розпізнавання випадків «зациклення» (deadlock) і перенаправлення одного з роботів на обхідний маршрут. Тому більшість підпунктів із секції навігації також виконані. Четверте, у блоці «Управління енергоспоживанням» система відстежує рівень заряду батареї кожного робота, за потреби роботи автоматично вирішують йти на зарядку і оцінюють, чи вистачить їм енергії для завершення нового замовлення перед його взяттям на опрацювання. По-п’яте, «Управління інвентарем» включає відстеження залишків товарів на палетах та полицях, автоматичне оновлення інформації про переміщення, резервування полиць під замовлення і забезпечення сумісності товарів на одній полиці — успішно реалізовано. Однак логіка перевірки товару та процесу реального опрацювання, потребує подальшої інтеграції у фізичний простір та дорозробки. Нарешті, у пункті «Збір статистики та звітність» система регулярно записує у звіт всю необхідну інформацію про симуляцію. Побудовано механізм формування звітів у вигляді текстового файлу, що дозволяє згодом аналізувати ефективність роботи кожного алгоритму та симуляції загалом; таким чином, і цей набір функцій реалізовано в повному обсязі. Що стосується «Інтеграції з базою даних», усі дані про стан складу, позиції інвентарю, статуси замовлень, інформація про роботів, полиці та палети зберігаються у реляційній БД на MS SQL Server. Операції зчитування і запису виконуються з використанням транзакцій, що забезпечує атомарність: у разі збою всі невдалі зміни автоматично відкидаються. Тому усі підпункти з інтеграції реалізовано.

Що ж стосується нефункціональних вимог (розділ 2.3.2), то тут результат виглядає таким чином. За показниками продуктивності обчислення маршруту на сітці розміром 20×41 займає в середньому близько 5 мс (залежно від завантаженості карти), що відповідає вимозі «не більше 100 мс». Система витримує одночасну роботу десяти роботів без помітної деградації у швидкості, час відповіді на запити до БД також повністю влаштовують систему. Deadlock-ситуації розв’язуються менше ніж за 10 с (зазвичай упродовж 2–3 с за рахунок перепланування маршрутів). Отже, усі вимоги з блоку «Продуктивність» виконано. З точки зору «Надійності» система коректно працює навіть у разі виходу з ладу одного з роботів. Можлива відновленість стану після аварійного завершення (за рахунок функцій транзакцій БД та журналювання подій). Щодо «Масштабованості», архітектура побудована на модульному підході: додавання нового робота не потребує змін коду основного ядра, а розширення системи новими модулями. Проте зміна розмірів складу вимагає коригування конфігурацій карт і часткової перегенерації маршрутних алгоритмів, що не цілком відповідає вихідній ідеї «без кардинальних змін». Щодо «Підтримуваності», код структуровано за явним розділенням на модулі: база даних, логіка навігації, диспетчер замовлень, енергоменеджмент, збір статистики. Система логування функціонує стабільно та надійно, тому в контексті подальшого вдосконалення та інтеграції реалізованої підсистеми не передбачається суттєвих труднощів. Нарешті, у питанні «Сумісності» підсистема на Windows з Python 3.12 і MS SQL Server 2019, працює без перебоїв, тож ця вимога задоволена.

Таким чином, переважна більшість функціональних і нефункціональних вимог була реалізована, однак існують окремі невиконані чи частково виконані вимоги, а також напрямки для подальшого вдосконалення:

1. Удосконалення логіки обробки замовлень, та впровадження приорітетних та не пріорітетних замовлень для роботів, але це потребує інтеграції з системою доставки;
2. Поліпшення алгоритму обробки складних deadlock-сценаріїв. Зараз базова модель дозволяє розпізнавати «зациклення» й аварійно «відхиляти» одного з агентів, але для великих скупчень роботів на вузьких ділянках корисно було б застосувати алгоритм «пошуку в просторі часу» (time-space A\*) або інтегроване резервування кількох кроків уперед. Це зменшить ймовірність невиправдано довгих очікувань та підвищить загальну ефективність при щільному навантаженні;
3. Розширення системи управління інвентарем щодо сумісності товарів. Наразі враховується лише базова сумісність (товари з одного замовлення), але не аналізуються додаткові атрибути (розмір, вага, характеристики зберігання). Для складських площ, де товари можуть мати різні температурні режими чи особливі вимоги до умови зберігання, доцільно додати механізми класифікації та додаткової перевірки перед розміщенням;
4. Повна реалізація масштабованості картографії. З метою виконати повноцінно вимогу «зміни розмірів складу без кардинальних змін» потрібна система автоматичного змінювання параметрів і конфігурацій при завантаженні різних розмірів сітки, а також перевірки валідності маршрутів. Пропонується інтегрувати «смарт-систему», яка автоматично пристосовуються до вхідних розмірів та обмежень нових зон, без жорсткого кодування розмірів та позицій на складі;
5. Також пропонується вдосконалити адміністративну панель складу шляхом розширення аналітичного функціоналу та надання адміністратору можливості вносити зміни до параметрів системи, вимикати та вмикати роботів віддалено. Але ці покращення пропунуються вже при інтеграції з реальним фізичним складом;
6. Рекомендується подальша інтеграція з штучним ітелектом, дозволить цій підсистемі прйимати більш правильні рішення с точки зору планування маршруту та оцінка пріорітетності замовлення.
7. Для покращення функції уникнення колізій, та успішної інтеграції у фізичний простір складу, підсистема потребує обов’язковой інтеграції з лідарами та датчиками руху.

Підсумовуючи, розроблена система у рамках дипломної роботи у значущій мірі відповідає заданим функціональним та нефункціональним вимогам. Водночас, для доведення її до виробничого рівня необхідні доопрацювання з боку вдосконалення навігаційних алгоритмів (особливо в умовах надвисокої щільності роботів), поліпшення логіки управління енергоспоживанням, розширення можливостей інвентаризації з урахуванням додаткових атрибутів товарів, а також повна реконфігурованість карти без залучення розробника. Завершення зазначених покращень дозволить зробити систему надійнішою, масштабованішою й більш універсальною для застосування на реальних об’єктах зі складною інфраструктурою і динамічними умовами роботи.

# **ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ**

[1] — [https://www.globenewswire.com/news—release/2024/09/26/2953889/0/en/Autonomous—Mobile—Robots—for—Logistics—and—Warehousing—Market—Set—to—Achieve—USD—18—56—Billion—by—2032—Driven—by—Growing—Automation—Demand.html](https://www.globenewswire.com/news-release/2024/09/26/2953889/0/en/Autonomous-Mobile-Robots-for-Logistics-and-Warehousing-Market-Set-to-Achieve-USD-18-56-Billion-by-2032-Driven-by-Growing-Automation-Demand.html%20)

[2] — [https://www.exotec.com/insights/how—amazon—robotics—has—changed—the—landscape—of—fulfillment/](https://www.exotec.com/insights/how-amazon-robotics-has-changed-the-landscape-of-fulfillment)

[3] — [https://www.aboutamazon.com/news/operations/how—amazon—deploys—robots—in—its—operations—facilities](https://www.aboutamazon.com/news/operations/how-amazon-deploys-robots-in-its-operations-facilities)

[4] — Кощій О. І. Склад і транспорт у логістиці: Навчальний посібник. Луцький національний технічний університет, 2020. URL: <https://elib.lntu.edu.ua/sites/default/files/elib_upload/Кощій/other/tema_11__sklad_i_transport_u_logisticzi.pdf>

[5] — Cкладська логістика : навчальний посібник / В. Є. Марчук, М. Ю. Григорак, О. М. Гармаш, О. В. Овдієнко. Київ: ОЛДІ—ПЛЮС, 2020. 256

<https://www.researchgate.net/publication/377659147_Skladska_logistika_navcalnij_posibnik>

[6] — [https://www.lucasware.com/the—roi—of—autonomous—mobile—robots—in—your—dc](https://www.lucasware.com/the-roi-of-autonomous-mobile-robots-in-your-dc)

[7] — [https://www.nature.com/articles/s41598—022—17684—0](https://www.nature.com/articles/s41598-022-17684-0)

[8] — Програмна інженерія підручник К. М. Лавріщева [https://csc.knu.ua/en/library/books/lavrishcheva—6.pdf](https://csc.knu.ua/en/library/books/lavrishcheva-6.pdf)

[9] — <https://docs.python.org/3/library/threading.html>

[10]—<https://uk.wikipedia.org/wiki/%D0%90%D1%82%D0%BE%D0%BC%D0%B0%D1%80%D0%BD%D1%96%D1%81%D1%82%D1%8C_(%D1%81%D0%B8%D1%81%D1%82%D0%B5%D0%BC%D0%B8_%D0%B1%D0%B0%D0%B7%D0%B8_%D0%B4%D0%B0%D0%BD%D0%B8%D1%85)>

[11]— [https://uk.wikipedia.org/wiki/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%BD%D1%96%D1%81%D1%82%D1%8C\_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)#%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%B5\_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F](https://uk.wikipedia.org/wiki/%D0%97%D0%B2%27%D1%8F%D0%B7%D0%BD%D1%96%D1%81%D1%82%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)%23%D0%9F%D1%80%D0%BE%D1%86%D0%B5%D0%B4%D1%83%D1%80%D0%BD%D0%B5_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D1%83%D0%B2%D0%B0%D0%BD%D0%BD%D1%8F)

[12] — <https://uk.wikipedia.org/wiki/%D0%95%D0%B2%D0%BA%D0%BB%D1%96%D0%B4%D0%BE%D0%B2%D0%B0_%D0%B2%D1%96%D0%B4%D1%81%D1%82%D0%B0%D0%BD%D1%8C>

[13] — <https://uk.wikipedia.org/wiki/%D0%9C%D0%B0%D0%BD%D0%B3%D0%B5%D1%82%D1%82%D0%B5%D0%BD%D1%81%D1%8C%D0%BA%D0%B0_%D0%BC%D0%B5%D1%82%D1%80%D0%B8%D0%BA%D0%B0>

[14] — [https://www.geeksforgeeks.org/dijkstras—shortest—path—algorithm—greedy—algo—7/](https://www.geeksforgeeks.org/dijkstras-shortest-path-algorithm-greedy-algo-7/)

[15] — <https://docs.python.org/3/library/math.html>

[16] — <https://docs.python.org/3/library/heapq.html>

# **ДОДАТОК А**

Тут буде Код програми

# **ДОДАТОК Б**

Тут буде «псевдо код»