Міністерство освіти і науки України
Рівненський державний гуманітарний університет
Кафедра інформаційних технологій та моделювання

Кваліфікаційна робота

за освітнім ступенем «магістр»

на тему:

МОДЕЛЮВАННЯ РОБОТИ СИСТЕМИ УПРАВЛІННЯ АВТОМАТИЗОВАНОГО СКЛАДУ

Виконав:
здобувач 2 курсу
групи М-КН-21
спеціальності 122 «Комп’ютерні науки»
Доронін Віталій Олегович

Науковий керівник:
к.ю.н., доцент Кіндрат П. В.

Рівне – 2025

АНОТАЦІЯ

Доронін В.О. Моделювання роботи системи управління автоматизованого складу. – Кваліфікаційна робота на правах рукопису.

Кваліфікаційна (магістерська) робота на здобуття ступеня магістра за спеціальністю 122 Комп'ютерні науки. – Рівненський державний гуманітарний університет. Рівне, 2025.

Кваліфікаційна (магістерська) робота присвячена дослідженню та розробці імітаційної моделі функціонування системи управління автоматизованим складом, що використовує роботизовані візки (AGV) для транспортування вантажів. Актуальність роботи зумовлена необхідністю оптимізації складських процесів та обґрунтування конфігурації автоматизованих систем в умовах зростання вимог до швидкості обробки замовлень.

У роботі спроектовано та реалізовано дискретно-подійну модель (Discrete Event Simulation), яка відтворює процеси надходження замовлень, формування черг, відбору товарів та їх транспортування. Програмна реалізація виконана мовою Python з використанням бібліотек SimPy для моделювання подій, NetworkX для побудови топології складу у вигляді графа та OR-Tools для вирішення задачі оптимального призначення завдань роботам.

Модель дозволяє проводити експериментальні дослідження впливу різних факторів (кількості AGV, політик диспетчеризації, стратегій зберігання товарів та профілів пікового навантаження) на ключові показники ефективності (KPI), такі як середній час циклу замовлення, пропускна здатність та довжина черг. Результати моделювання доводять, що використання оптимізаційних алгоритмів диспетчеризації та правильне налаштування ресурсів значно підвищують ефективність роботи складу в умовах нерівномірного попиту. Розроблений програмний комплекс має практичну цінність як інструмент підтримки прийняття рішень при проектуванні та експлуатації логістичних систем.

Ключові слова: Автоматизований склад, Система управління складом (WMS), Транспортні роботи (AGV), Імітаційне моделювання, Дискретно-подієва модель, Диспетчеризація, Задача призначення, SimPy, Логістика.

ABSTRACT

Doronin V.O. Modeling the operation of an automated warehouse management system. – Qualification paper as a manuscript.

Qualification (Master's) thesis for the degree of Master in specialty 122 Computer Sciences. – Rivne State University of Humanities. Rivne, 2025.

The qualification (Master's) thesis is devoted to the research and development of a simulation model for the functioning of an automated warehouse management system that uses automated guided vehicles (AGVs) for cargo transportation.

The relevance of the work is due to the need to optimize warehouse processes and substantiate the configuration of automated systems given the increasing requirements for order processing speed.

The work designs and implements a discrete-event model (Discrete Event Simulation), which reproduces the processes of order arrival, queue formation, goods picking, and their transportation.

The software implementation is performed in Python using SimPy libraries for event modeling, NetworkX for building the warehouse topology as a graph, and OR-Tools for solving the problem of optimal task assignment to robots.

The model allows for experimental studies of the influence of various factors (number of AGVs, dispatching policies, goods storage strategies, and peak load profiles) on key performance indicators (KPIs), such as average order cycle time, throughput, and queue length.

Simulation results prove that the use of optimization algorithms for dispatching and proper resource configuration significantly increase the efficiency of warehouse operations under conditions of uneven demand.

The developed software complex has practical value as a decision support tool in the design and operation of logistics systems.

Keywords: Automated Warehouse, Warehouse Management System (WMS), Automated Guided Vehicles (AGV), Simulation Modeling, Discrete Event Simulation (DES), Dispatching, Assignment Problem, SimPy, Logistics.

ЗМІСТ

ВСТУП

Розвиток електронної комерції, сервісів «доставка за день» та глобальних логістичних ланцюгів призводить до різкого зростання вимог до швидкості, надійності й передбачуваності роботи складів. Сучасні розподільчі центри повинні обробляти тисячі замовлень на годину, адаптуватися до пікових навантажень та забезпечувати мінімальний час циклу замовлення. За таких умов традиційні склади з ручним відбором і транспортуванням вантажів дедалі частіше виявляються неефективними, що зумовлює перехід до автоматизованих рішень на основі роботизованих візків (AGV – Automated Guided Vehicles) та систем управління складом (WMS – Warehouse Management System).

Одним із ключових викликів під час впровадження автоматизованих складів є обґрунтування конфігурації системи: кількості транспортних роботів, їхніх характеристик, політик диспетчеризації, топології складу, стратегій розміщення товарів, а також налаштування режимів роботи під нерівномірний попит. Прийняття таких рішень лише на основі евристики або інтуїції може призвести до недовикористання дорогого обладнання або, навпаки, до його перевантаження, зростання черг та часу обслуговування.

Імітаційне моделювання, зокрема дискретно-подійні моделі, дають можливість без ризику та додаткових витрат «програти» різні сценарії функціонування автоматизованого складу, оцінити ключові показники ефективності (KPI), знайти вузькі місця в системі та перевірити, як змінюється продуктивність за умов пікових навантажень, відмов обладнання, зміни політики керування тощо. Саме тому розроблення й дослідження моделі роботи системи управління автоматизованого складу є актуальним науково-практичним завданням.

У даній роботі розглядається підхід до моделювання потоку замовлень, мережі черг, роботи парку транспортних роботів та алгоритмів їх диспетчеризації з використанням засобів дискретно-подійного моделювання та математичної

оптимізації. Модель дозволяє досліджувати вплив різних факторів – кількості AGV, профілю попиту, політики призначення завдань, стратегії зберігання товарів, режиму відбору – на такі KPI, як середній час циклу замовлення, довжина черг, завантаженість роботів, частка невиконаних замовлень тощо.

Об'єкт дослідження – процес функціонування системи управління автоматизованим складом із використанням транспортних роботів (AGV) для обробки потоку замовлень.

Предмет дослідження – імітаційна модель системи управління автоматизованим складом, її структура, алгоритми диспетчеризації AGV, моделі попиту та мережі черг, а також програмні засоби для реалізації та аналізу роботи такої моделі.

Мета дослідження – розробити та дослідити дискретно-подійну імітаційну модель системи управління автоматизованим складом із парком транспортних роботів, яка дозволяє аналізувати вплив конфігурації системи та профілю навантаження на показники ефективності й підтримувати прийняття рішень щодо параметрів та політик роботи складу.

Для досягнення поставленої мети поставлено такі основні завдання:

- проаналізувати сучасні підходи до автоматизації складів, систем управління складом (WMS) та імітаційного моделювання логістичних процесів;
- сформувати вимоги до моделі автоматизованого складу, визначити необхідні вхідні параметри та ключові показники ефективності;
- спроектувати логічну та програмну архітектуру імітаційної моделі, включаючи модулі генерації попиту, моделювання мережі черг, руху AGV та алгоритмів їх диспетчеризації;
- реалізувати дискретно-подійну модель роботи складу із застосуванням відповідних бібліотек (SimPy, OR-Tools, засобів аналізу даних та візуалізації) та забезпечити збір телеметрії й розрахунок KPI;

– виконати верифікацію та валідацію моделі на основі юніт-тестів, балансових перевірок та зіставлення з аналітичними оцінками для спрощених сценаріїв;

– розробити план експериментів і провести серію симуляцій для різних сценаріїв (різна кількість AGV, політики диспетчеризації, профілі попиту, стратегії зберігання та режими відбору);

– проаналізувати результати експериментів, виявити ключові фактори, що впливають на ефективність роботи складу, та сформулювати практичні рекомендації щодо налаштування системи.

Методи дослідження базуються на поєднанні теорії масового обслуговування, дискретно-подійного імітаційного моделювання, методів стохастичного моделювання потоків подій, а також методів математичної оптимізації для розв’язання задачі призначення (зокрема, із використанням бібліотеки OR-Tools). Для обробки та візуалізації результатів застосовано інструменти аналізу даних (табличні структури, статистичні показники, побудова графіків).

Практична значущість роботи полягає в тому, що розроблена імітаційна модель та програмний комплекс можуть бути використані:

– для попереднього обґрунтування конфігурації автоматизованого складу (кількість і параметри транспортних роботів, вибір політики диспетчеризації, стратегії зберігання товарів);

– для аналізу стійкості системи до пікових навантажень та відмов обладнання;

– як навчальний інструмент під час вивчення дисциплін, пов’язаних з логістикою, моделюванням систем та оптимізацією, оскільки дозволяє наочно демонструвати роботу мережі черг та вплив керуючих рішень на результат.

Таким чином, представлена в роботі модель спрямована на підвищення якості проектування та експлуатаційних рішень для автоматизованих складів, що є

актуальним завданням в умовах цифровізації логістичних процесів та зростання конкуренції на ринку складських послуг.

РОЗДІЛ 1.
ТЕОРЕТИЧНЕ ПІДҐРУНТЯ ФУНКЦІОНУВАННЯ АВТОМАТИЗОВАНОГО СКЛАДУ

1.1 Автоматизовані склади та класифікація систем

Автоматизований склад – це комплекс технологій, що мінімізує роль людини у зберіганні й переміщенні товарів. Основні типи таких систем включають AS/RS, шатл-системи, AGV та AMR, кожна з власними особливостями та перевагами.

– **AS/RS (Automated Storage and Retrieval Systems)**: механізовані склади, що автоматично розміщують і видають товари зі стелажів. Вони поєднують робототехніку, конвеєри та ліфти, усуваючи необхідність ручного пошуку товарів, підвищуючи щільність зберігання та зменшуючи помилки при комплектації замовлень[1]. Впровадження AS/RS дозволяє значно скоротити трудові витрати і оптимально використовувати складський простір, знижуючи навіть екологічний вплив[2].

– **Шатл-системи**: різновид AS/RS з багаторівневими стелажами, де спеціальні шатли рухаються по рівнях і транспортують контейнери або коробки до ліфтів чи конвеєрів. Шатл-системи ефективні для дрібних вантажів і забезпечують високий паралелізм роботи, завдяки чому продуктивність складу вища порівняно з традиційними системами з одним краном-штабелером.

– **AGV (Automated Guided Vehicles)**: автоматизовані транспортні засоби, що рухаються заданими маршрутами за допомогою датчиків (лазерних, магнітних, лідара) і розмітки на підлозі[3]. Вони надійні для повторюваних завдань, як-от перевезення піддонів між зонами складу, подача компонентів на виробничі лінії тощо[4]. AGV бувають різних типів (вилочні навантажувачі, буксири, платформні роботи) та автоматизують внутрішньоскладське транспортування, усуваючи потребу в ручному керуванні технікою.

– **AMR (Autonomous Mobile Robots)**: автономні мобільні роботи, оснащені камерами та сенсорами. На відміну від AGV, вони здатні динамічно змінювати маршрут при появі перешкод[5]. Якщо шлях заблокований, AMR об'їде перешкоду й оптимізує маршрут у реальному часі[5]. Такі роботи можуть виконувати ширший спектр задач – від перевезення вантажів до автономного відбору дрібних товарів[6] – і не потребують фізичної інфраструктури (маршрути задаються цифровою картою), що спрощує їх розгортання. AMR гнучкі до перепланування складу та легко масштабуються шляхом додавання нових роботів.

Переваги автоматизації: перелічені системи доповнюють одна одну в логістичному ланцюгу. AS/RS і шатл-системи реалізують підхід "goods-to-person" – товари автоматично доставляються до працівника, пришвидшуючи комплектацію замовлень. AGV та AMR забезпечують переміщення товарів між зонами (приймання, зберігання, відвантаження), створюючи безперервний потік з мінімальними затримками. Загалом автоматизація складу підвищує продуктивність, скорочує час обробки замовлень, підвищує точність і покращує безпеку, адже важку фізичну роботу виконують роботи[7]. Такі інвестиції особливо ефективні на середніх і великих складах із високою інтенсивністю операцій, де швидкість і точність виконання замовлень критично впливають на рівень сервісу.

1.2 Функції WMS та політики управління

Ефективне функціонування автоматизованого складу неможливе без системи управління складом (WMS, Warehouse Management System). WMS – це програмне забезпечення, яке централізовано контролює всі складські операції: від надходження товарів до їх відвантаження клієнтам[8]. Основна мета WMS – оптимізація роботи складу через автоматизацію і координацію дій персоналу та обладнання. Система виконує приймання товару, визначає місця його розміщення; керує адресним зберіганням, поповненням запасів; забезпечує комплектацію замовлень і відвантаження з оформленням документів[8]. WMS також веде облік товарів у

реальному часі та аналізує показники роботи (швидкість обробки, продуктивність персоналу тощо)[8]. Впровадження WMS дозволяє організувати адресне зберігання: кожна товарна позиція закріплена за конкретною коміркою, що усуває хаотичність і скорочує час пошуку товарів працівниками [9].

WMS реалізує різні **політики управління** складськими процесами – визначає, як система розміщує товари, як формує та випускає замовлення на відбір, і як призначає ці завдання ресурсам. Основні типи таких політик:

1. **Політики зберігання товарів** – визначають принцип розміщення SKU по комірках складу. Закріплене зберігання (dedicated storage) передбачає фіксоване місце для кожного SKU. Це спрощує пошук товару, але простір використовується неефективно, бо місце зарезервовано навіть коли запас цього SKU мінімальний[10][11]. Випадкове зберігання (random storage) не закріплює фіксованих місць: товар розміщується в будь-яку доступну комірку. Такий підхід максимально ефективно використовує простір, проте ускладнює пошук і може збільшувати відстані переміщень при відборі товарів[12]. Кластеризоване зберігання (class-based storage) – компромісний підхід: товари діляться на класи A/B/C за частотою попиту, і кожному класу виділена своя зона на складі[13][14]. Найбільш популярні SKU класу A (приблизно 15% номенклатури, що забезпечують ~80% обороту) зберігаються ближче до зони відбору, менш ходові (клас C) – далі[15]. Це скорочує відстані переміщення під час відбору популярних товарів[15], хоча потребує резервування площ під кожен клас, що дещо знижує загальну щільність зберігання. Дослідження підтверджують, що правильно впроваджені класові або COI-методи розміщення (Cube-per-Order Index) помітно скорочують пробіг пікерів порівняно з випадковим зберіганням[16].

2. **Політики відбору замовлень** – визначають, як формуються і запускаються замовлення на комплектацію.

– Безперервний відбір означає, що замовлення надходять у роботу відразу після появи, без затримок. Такий режим характерний для динамічних e-commerce складів, де важлива мінімальна затримка виконання замовлення[17].

– Відбір хвилями групує замовлення у хвилі, які запускаються в заплановані інтервали або за певними критеріями. Під час хвилі пікери обробляють групу замовлень разом, що може підвищити ефективність (менше повторних обходів тих самих локацій), але між хвилями виникають паузи, поки не почнеться наступна хвиля[17][18]. Сучасні WMS підтримують також накладання хвиль і динамічну пріоритизацію, коли хвилі частково перекриваються, дозволяючи паралельно завершувати одну і починати наступну – це наближає процес до безперервного режиму[19].

Вибір між безперервним відбором і хвилями залежить від профілю роботи складу.

3. **Політики диспетчеризації завдань** – регулюють розподіл робіт між ресурсами (працівниками чи роботами) при наявності черги задач. Коли у системі накопичується багато запитів, WMS повинна вирішити, яке завдання виконати першим і кому його доручити. Найпоширеніше правило – FIFO (First In, First Out), тобто виконувати завдання в порядку їх надходження. Інші правила враховують характеристики завдань: SJF (Shortest Job First) надає пріоритет найкоротшим задачам (щоб мінімізувати середній час виконання), проте ризикує відкладати довгі завдання. NJF (Nearest Job First) часто застосовують для рухомих роботів: першим береться завдання, найближче до поточного місця знаходження ресурсу (AGV/пікера), щоб скоротити холості пробіги. Такий підхід підвищує ефективність використання ресурсу, але потребує актуальної оцінки відстаней у реальному часі. Також використовують правила за терміном виконання (Earliest Due Date), коли першочергово виконуються замовлення з найближчим дедлайном відправки – це важливо для забезпечення рівня сервісу (максимізації частки замовлень, відправлених вчасно). Нерідко в WMS комбінують кілька критеріїв: наприклад,

спочатку задачі сортуються за критичністю дедлайну, а серед задач з однаковим пріоритетом – за FIFO. Для флоту AGV вибір правильної політики диспетчеризації є частиною управління парком роботів і може суттєво впливати на показники системи.

Таким чином, WMS виступає “мозком” автоматизованого складу, реалізуючи стратегії зберігання товарів, випуску замовлень та оптимального завантаження ресурсів. Гнучкість налаштувань WMS дозволяє підібрати політики під конкретний профіль роботи.

1.3 Моделі попиту і черг на складі

Операційна робота складу має випадковий характер: надходження замовлень від клієнтів у часі, поява задач на поповнення запасів, прибуття вантажів на приймання тощо. Для опису цих процесів широко застосовують стохастичні моделі, зокрема пуассонівські процеси. Пуассонівський процес характеризується середньою інтенсивністю подій *lambda* та властивостями стаціонарності й відсутності післядії: імовірність появи *k* подій за інтервал *t* описується розподілом Пуассона *P{N(t)=k} = frac{(lambda t)^k e^{-lambda t}}{k!},* а інтервали між послідовними подіями мають експоненційний розподіл з параметром *lambda*[20][21]. Для складу це означає, наприклад, що замовлення надходять з постійною середньою швидкістю *lambda*, незалежно одне від одного: більшість інтервалів між замовленнями короткі, але іноді трапляються й довгі паузи (що зумовлено властивостями експоненційного розподілу).

У реальності інтенсивність надходження замовлень може змінюватися протягом доби. Для таких випадків використовують неоднорідний пуассонівський процес (NHPP), в якому інтенсивність *lambda(t)* є функцією часу. Наприклад, можна задати добовий цикл: вранці *lambda(t)* висока (багато замовлень), вночі майже нульова. NHPP зі змінною інтенсивністю зберігає основну властивість: кількість подій у будь-якому інтервалі *[t_1, t_2]* має пуассонівський розподіл з параметром

Lambda = int_{t_1}^{t_2} lambda(t)dt. Це зручно для моделювання сезонності або добових коливань попиту.

При симуляції роботи складу важливо правильно врахувати накопичення черги замовлень. Зазвичай припускають, що потік заявок (замовлень) можна моделювати пуассонівським процесом. Далі, склад з обмеженою кількістю ресурсів (пікерів, пакувальників, роботів) можна розглядати як систему масового обслуговування, де замовлення очікують у черзі, поки їх не почнуть обробляти. Структура черги на складі часто багатостадійна: наприклад, спочатку замовлення може чекати черги на відбір товарів (якщо всі пікери зайняті), потім – на пакування, далі – на відвантаження. Кожну стадію можна описати окремою лінією обслуговування із певним розподілом часу операції. В аналітичних моделях використовують нотацію типу *GI/G/c* (загальні довільні розподіли інтервалів між подіями та часів обслуговування, c – кількість паралельних каналів), але через складність таких систем частіше звертаються до імітаційного моделювання (дискретно-подієвого), ніж до чисто аналітичних розрахунків.

На початку симуляції система може перебувати в штучному стані (наприклад, на час *t=0* усі склади порожні, черги немає, ресурси вільні), який не відповідає усталеному режиму. Якщо одразу збирати статистику, вона буде спотворена цими початковими умовами. Тому вводять період прогріву моделі (warm-up) – початковий відрізок часу симуляції, протягом якого зібрані дані не враховуються у фінальних результатах. Це дає системі змогу "розігрітися" та наблизитися до стаціонарного режиму. Існують емпіричні та статистичні методи визначення тривалості періоду прогріву: часто моделюють систему кілька разів із різними *T_0* і спостерігають, з якого моменту ключові показники (наприклад, середня довжина черги, завантаження ресурсів) стабілізуються в межах довірчих інтервалів. Для нашої моделі можна обрати *T_0*, достатній щоб заповнити склад до робочого рівня запасів і сформувалася репрезентативна черга замовлень перед збором статистики.

У стохастичних моделях складу черга замовлень наростає, коли інтенсивність надходження перевищує пропускну здатність системи. Якщо, наприклад, замовлення надходять зі швидкістю 20 на годину, а пікери встигають обробити лише 15 на годину, то черга буде зростати. З іншого боку, якщо ресурси здатні обробляти більше, ніж надходить (тобто система має запас продуктивності), черга залишатиметься малою, а час очікування замовлень – коротким. У моделюванні ми фіксуємо такі показники, як середня довжина черги та середній час очікування замовлення – ці метрики допомагають оцінити якість обслуговування та завантаженість системи. Особливо важливо досліджувати черги в сценаріях пікового навантаження, щоб зрозуміти, чи зможе склад впоратися з підвищеним попитом без накопичення значних відставань у виконанні замовлень.

Отже, пуассонівські моделі надходження заявок та теорія черг є базовими інструментами для представлення випадкового характеру попиту на складські послуги. У подальшому, при побудові імітаційної моделі, ми будемо використовувати ці підходи для генерації подій (нових замовлень) та відстеження черг на різних етапах процесу обробки замовлень.

1.4 Дискретно-подієве моделювання складу

Для аналізу складних систем із випадковою динамікою, таких як складські операції, використовують дискретно-подієве моделювання (Discrete Event Simulation, DES). DES представляє роботу системи як послідовність окремих подій у часі: між подіями стан системи не змінюється, всі ключові зміни відбуваються миттєво у моменти подій[22]. Це дозволяє симулятору "перестрибувати" через періоди бездіяльності одразу до часу наступної події, що суттєво підвищує ефективність моделювання[23].

У DES-моделі складу можна виділити кілька основних компонентів:

– **Подія** – миттєва зміна стану системи в певний час t. У контексті складу подіями можуть бути: прибуття нового замовлення, початок відбору товару пікером,

завершення операції, вихід замовлення зі складу тощо. Кожна подія впливає на систему: змінює значення змінних стану (наприклад, подія "нове замовлення" збільшує довжину черги замовлень).

– **Стан системи** – сукупність змінних, що описують систему у будь-який момент. Для складу такими змінними є, наприклад, кількість замовлень у черзі на відбір, місцеположення кожного AGV, залишки товарів на полицях, стан кожного пікера (вільний чи зайнятий) тощо. Події змінюють стан: наприклад, подія “нове замовлення” збільшує довжину черги, а подія “завершення відбору” звільняє пікера (зменшує число зайнятих каналів ресурсу).

– **Ресурси** – об’єкти з обмеженою пропускною здатністю, які обслуговують процеси. У складі ресурсами є пікери (людські працівники або роботизовані станції відбору), пакувальні автомати, ліфти шатл-систем, самі шатли чи AGV. Ресурс має певну кількість каналів (паралельних серверів): наприклад, один пікер може одночасно працювати лише над 1 замовленням (місткість ресурсу = 1); якщо пікерів 5, маємо 5 каналів. Процеси затримуються в черзі при ресурсі, якщо всі канали зайняті.

– **Процес** – логічна схема або алгоритм дій, що описує поведінку окремого активного елемента чи транзакції в системі. У процесно-орієнтованому підході (як у бібліотеці SimPy чи мові GPSS) моделюють, наприклад, процес **Order** (замовлення): воно прибуває, стає в чергу до пікера, чекає, потім виконується відбір, далі переходить на пакування тощо. Кожен такий процес "живе" окремо, генеруючи події (початок і кінець обслуговування тощо). Інший приклад – процес **AGV**, який отримує завдання, рухається по графу до точки, чекає на завантаження, везе вантаж до зони відправки. DES дозволяє описати ці процеси у вигляді коду: процес призупиняється, коли ресурс недоступний (чекає події звільнення ресурсу), і відновлюється при настанні потрібної події. Такий підхід називають симуляцією на основі процесів.

– **Глобальний час** – поточний модельний час. Рушій дискретно-подієвого моделювання підтримує впорядковану чергу запланованих подій. Він бере найближчу майбутню подію, просуває глобальний час до моменту її настання, оновлює стан системи і виконує пов’язані з подією дії (викликає обробники події). Потім переходить до наступної події. Таким чином симуляція імітує хід часу стрибками від події до події[23].

Для реалізації DES існує багато інструментів. У Python одним із популярних є бібліотека **SimPy** – вона надає класи Environment (середовище моделювання з часовим рушієм), Process (для опису процесів на основі генераторів Python), Resource (для моделювання обмежених ресурсів). У нашому проєкті симуляції складу використано SimPy для створення процесів замовлень, процесів руху AGV тощо. DES дозволяє детально змоделювати взаємодію компонентів: наприклад, якщо AGV прибув до ліфта шатл-системи і чекає, поки ліфт звільниться – це відображено як очікування ресурсу; поки ліфт зайнятий, події AGV не відбуваються, а глобальний час може перейти до інших процесів.

Дискретно-подієве моделювання є адекватним підходом для систем типу "склад", де ключові операції дискретні і утворюються черги. DES забезпечує точність та гнучкість моделювання різних правил і сценаріїв, хоча може вимагати значних обчислювальних ресурсів при великих масштабах або тривалому періоді симуляції. Наш симулятор на Python реалізує DES-підхід, що дозволяє експериментувати з різними політиками WMS (описаними вище в розд. 1.2) та оцінювати їх вплив на показники складу.

1.5 Маршрутизація на графі складу

В автоматизованому складі з мобільними роботами важливим завданням є пошук оптимальних маршрутів руху роботів від однієї точки до іншої. План складу представляють у вигляді графа: вузли відповідають значущим локаціям (перехрестя коридорів, точки відбору/відправки, зарядні станції, парковки), а ребра – можливим

шляхам між цими локаціями[24]. Кожному ребру призначається вага – наприклад, час руху по цьому сегменту або відстань.

Для знаходження найкоротшого шляху на такому графі (за критерієм мінімального часу або відстані) використовується алгоритм Дейкстри[25][26]. Для позитивно зважених графів він гарантує знаходження оптимального маршруту від заданого стартового вузла до всіх інших, а його обчислювальна складність $O((N+E)\log N)$ є прийнятною, адже розміри складських графів помірні (порядку сотень вузлів). У нашій моделі кожне переміщення AGV планується саме як найкоротший шлях на графі складу.

Вага ребер може залежати від швидкості AGV. Якщо критерієм є час, то вага пропорційна довжині сегменту та швидкості руху робота: припустімо, AGV їздить всюди з однаковою максимальною швидкістю v (скажімо, 1.5 м/с), тоді ребро довжиною d має вагу (час) $t = d/v$. Якщо швидкість однакова, мінімальний час шляху збігається з мінімальною відстанню. Проте в реальних умовах швидкість може залежати від обставин: наприклад, на прямих ділянках робот їде швидко (1.5 м/с), на поворотах чи у вузьких проходах – повільніше (0.5 м/с), а з вантажем – теж обмежено (скажімо 1.0 м/с). Це можна врахувати, задавши ребрам різні ваги для різних станів робота або вводячи динамічну вагу: час на ребрі = d / v, де v – допустима швидкість робота на даному сегменті з урахуванням його стану (порожній/завантажений, тип ділянки тощо).

Граф складу зазвичай нагадує сітку коридорів між стелажами. Деякі маршрути можуть бути односторонніми (щоб уникнути зустрічного руху в вузьких проходах) – це моделюється як орієнтовані ребра. Також у граф можуть бути включені вузли парковки чи буферні зони – спеціальні місця, де AGV можуть чекати завдання, не блокуючи основні проходи.

Після знаходження найкоротшого шляху AGV дотримується його під час руху. У нашій моделі алгоритм Дейкстри використовується для статичного планування маршруту. У динамічному середовищі, де на шляху можуть з’являтися інші AGV або

непередбачені перешкоди, може знадобитися динамічне перепланування: робот може періодично оновлювати "карту" (наприклад, відзначаючи зайняті іншими AGV сегменти) та перевиконувати пошук шляху (можливо, за алгоритмом A* для більшої швидкості пошуку з евристикою). Але базовий рівень – це саме пошук найкоротшого шляху на статичному графі, що ми й реалізуємо в моделі.

У симуляції граф використовується двояко. По-перше, для обчислення відстаней і часу переміщення – коли диспетчеру (розд. 1.6) потрібно оцінити, скільки часу займе роботам під'їхати до певної локації, він виконує пошук шляху і множить довжину на швидкість. По-друге, для маршрутизації під час симуляції – процес руху AGV можна моделювати як низку подій із затримками, рівними часу переходу по кожному ребру. У спрощеному варіанті можна не анімувати кожен поворот, а відразу розрахувати сумарний маршрут і поставити подію "AGV прибув" через певний час. Якщо ж моделюється одночасно кілька AGV і можливі конфлікти на дорогах, тоді доводиться деталізувати рух: вводити події входу на ребро та виходу з нього, слідкувати за зайнятістю сегментів (щоб уникнути зіткнень) – це значно ускладнює модель і виходить за межі теоретичного огляду. Як постановка задачі, керування трафіком AGV потребує окремих алгоритмів розв'язання конфліктів, бронювання сегментів або динамічного перепланування маршрутів при взаємодії кількох роботів одночасно[27][28]. У нашій моделі припущено спрощено, що AGV не блокують один одного (наприклад, їх небагато або є достатньо об'їзних шляхів), тому задачі маршрутизації зводяться до незалежного планування шляху для кожного робота.

Таким чином, графова модель складу є основою для навігації роботів. Класичний алгоритм (Дейкстри або A*) дає змогу роботам їздити оптимальними маршрутами з точки зору часу або відстані, враховуючи їх швидкість. Це мінімізує витрати часу на логістичні переміщення всередині складу і підвищує загальний Throughput системи. У наступному розділі розглянемо, як на основі оціненого часу проїзду вирішується задача призначення завдань роботам.

1.6 Метрики ефективності та вартісна модель

Для оцінки роботи складу використовуються ключові показники ефективності (KPI). У моделі важливо визначити, які метрики збирати та як інтерпретувати їх з точки зору продуктивності і вартості:

– **Пропускна здатність** – кількість замовлень, виконаних за одиницю часу. Для складу її часто вимірюють як число виконаних замовлень на годину або за зміну. Це інтегральний показник продуктивності: наскільки багато замовлень склад може обробити. Через імітацію можна оцінити середню пропускну здатність, а також дослідити пікові значення[30]. Підвищення параметру свідчить про кращу ефективність процесів або збільшення ресурсів.

– **Час циклу замовлення** – повний час від надходження замовлення на склад до його відправлення (виконання). Часто його розглядають як загальний час перебування замовлення у системі. Це важливий показник з точки зору клієнтського сервісу: короткий – означає швидке виконання замовлення. У симуляції можна збирати як середній час циклу, так і розподіл. Для аналізу систем масового обслуговування відома спрощена формула Літтла: середня кількість замовлень у системі $L = lambda W$, де W – середній час перебування (lead time), λ – інтенсивність надходження замовлень. Це допомагає перевіряти консистентність результатів моделювання.

– **Рівень завантаження** ресурсів – частка часу, протягом якого ресурс зайнятий продуктивною роботою. Ми можемо виміряти його для кожного типу ресурсів: для AGV – відсоток часу, коли вони в русі чи виконують завдання (не простоюють), для пікерів – відсоток часу, коли вони відбирають товар (не очікують замовлень). Високе завантаження (наближення до 100%) може означати вузьке місце: ресурс працює на межі можливостей і не зможе обробити більший попит. Надто низьке (наприклад < 50%) означає, що ресурс недовикористаний – можливо, система має зайві потужності або процеси незбалансовані.

– **Експлуатаційні витрати (Opex)** – витрати на роботу складу. В моделі можна використати спрощену вартісну структуру: наприклад, вартість праці пікера (грн/год), вартість експлуатації AGV (електроенергія + амортизація, грн/год). Собівартість виконання замовлення розраховується як Opex, поділений на кількість виконаних замовлень[31]. Цей показник дає економічну оцінку ефективності: чим нижча вартість на замовлення, тим дешевше обходиться логістика для компанії. У симуляції можна для кожного сценарію обчислити середню вартість на одне замовлення. Наприклад, якщо 5 пікерів по 100 грн/год і 3 AGV по 50 грн/год працюють, і за годину виконано ~120 замовлень, то операційні витрати за годину ≈ 5 \ 100 + 3 \ 50 = 650 грн; отже cost per order ≈ 650/120 ≈ 5.4 грн/замовлення. Порівняння цього показника між різними сценаріями (наприклад, з різною кількістю AGV) покаже економічну доцільність додаткової автоматизації.

– **OTD (On-Time Delivery)** – відсоток замовлень, доставлених вчасно (у встановлений термін). Для внутрішнього складу це може означати частку замовлень, час циклу яких не перевищує певний поріг. Наприклад, якщо клієнтам обіцяна відправка протягом 2 годин з моменту отримання замовлення, то OTD = частка замовлень, виконаних ≤ 2 год. Цей показник відображає рівень сервісу та дотримання зобов'язань перед клієнтами. У моделі його легко розрахувати: для кожного замовлення фіксуємо час виконання і порівнюємо із заданим дедлайном.

– **Штрафи** – розширення вартісної моделі у випадку невиконання цілей сервісу. Якщо замовлення виконано із запізненням (пізніше обіцяного строку), можна додати умовний штраф P до загальних витрат. Таким чином моделюється упущена вигода або компенсація клієнту за прострочку. У симуляції це дозволяє кількісно оцінити компроміс між економією ресурсів і погіршенням сервісу: наприклад, зменшуючи кількість AGV, ми знижуємо прямі витрати, але дедлайни частіше порушуються і ростуть штрафи – сумарна "вартість на замовлення" може навіть зрости.

Можна відстежувати й інші показники, залежно від завдань аналізу: варіативність часу циклу, точність комплектування (% замовлень без помилок), пробіг пікерів за зміну, рівень заповнення складу, тощо. У межах даної роботи основна увага приділяється показникам продуктивності і базовій оцінці вартості.

1.7 Експериментальний дизайн симуляційного дослідження

Щоб результати моделювання були надійними та статистично обґрунтованими, необхідно грамотно спланувати експеримент. Це включає визначення числа повторів (реплікацій) симуляції, розрахунок довірчих інтервалів, вибір факторів і рівнів для порівняння альтернатив.

Реплікації. Оскільки моделювання містить випадкові величини, один прогін дає лише реалізацію стохастичного процесу. Для оцінки середніх значень показників та їх варіації потрібно виконати кілька незалежних запусків – реплікацій – моделі. Кожна реплікація використовує інше зерно генератора випадкових чисел, щоб забезпечити різні випадкові сценарії. Реплікацій має бути достатньо багато, щоб усереднити стохастичні флуктуації. Наприклад, можна провести 10 чи 20 запусків тривалістю по 8 годин кожен і зібрати по них необхідну статистику тощо.

Зерна випадковості (seed). Встановлення фіксованих seed для генераторів дає змогу зробити експерименти відтворюваними – той самий seed забезпечить ту саму послідовність випадкових подій. У дослідженні варто задати набір різних зерен (наприклад, 1, 2, 3, … для реплік) або генерувати їх випадково, щоб виключити кореляцію між реплікаціями. Іноді використовують метод спільного зерна для порівняння двох сценаріїв – запускають обидва з однаковими послідовностями випадковостей, щоб зменшити варіативність різниці.

Довірчі інтервали. Після виконання реплікацій для кожного показника можна обчислити оцінку середнього $\bar{X}$ і стандартну помилку $SE = s/\sqrt{r}$(де s – вибіркове стандартне відхилення з r реплік). На цій основі будується, наприклад, 95% довірчий інтервал: $\bar{X} \pm t_{0.975,\,r-1} \cdot SE$, де t – квантиль розподілу

Стьюдента. Такий інтервал з 95% ймовірністю містить істинне середнє значення показника[32]. Довірчі інтервали дають уявлення про статистичну значущість відмінностей: якщо інтервали двох сценаріїв не перекриваються, можна стверджувати, що різниця є значущою з обраним рівнем довіри. В симуляційному дослідженні зазвичай прагнуть отримати інтервали прийнятної ширини (наприклад, не більше ±5% від оцінки). Якщо інтервал надто широкий, збільшують число реплікацій.

Фактори і рівні. План експерименту визначає, які фактори (незалежні змінні) ми будемо варіювати, і які значення цих факторів перевіряти. У нашій задачі факторами можуть бути: кількість AGV на складі, кількість пікерів, політика відбору (хвильова vs безперервна), політика зберігання (random vs class-based) тощо. Кожен фактор може мати кілька рівнів: наприклад, для числа AGV – 2, 4, 6 штук; для політики – 2 типи. Якщо досліджують вплив одного фактору, можна фіксувати інші і змінювати лише цей один (однофакторний експеримент). Але часто цікавить комбінований вплив, тоді планують багатофакторний експеримент – з комбінаціями рівнів. Наприклад, 2 рівні фактору A (політика зберігання) × 2 рівні фактору B (політика відбору) дають 4 сценарії для симуляції. Повний факторіальний дизайн перевіряє всі можливі комбінації рівнів, що дає максимальну інформацію (можна оцінити і основні ефекти, і взаємодії факторів), але кількість сценаріїв зростає експоненціально з числом факторів. Тому інколи застосовують дробовий факторіальний план або інші методи DOE, щоб зменшити обсяг моделювання, зберігши основну інформацію.

Верифікація і валідація. Важливо перед проведенням фінальних експериментів переконатися, що модель коректно реалізує логіку (верифікація) і що результати мають сенс, узгоджуються з аналітичними оцінками чи реальними даними (валідація). Для цього роблять тестові запуски зі спрощеними параметрами, порівнюють із розрахунками (наприклад, для системи M/M/1 відомі формули для

середньої довжини черги – можна налаштувати симуляцію у відповідний режим і звірити результат).

РОЗДІЛ 2.
АНАЛІЗ І ПРОЄКТУВАННЯ СИСТЕМИ

2.1 Постановка задачі

Метою моделювання є дослідження процесу обробки замовлень на складі із використанням автоматизованих транспортних роботів (AGV) для доставки товарів. Система моделює весь цикл виконання замовлення від моменту його надходження до видачі клієнту. До меж моделі входять процеси надходження замовлень, відбір товарів зі складу та транспортування їх до зони відправки за допомогою AGV. Зовнішні процеси, такі як поповнення складу або зовнішня логістика, не розглядаються.

При моделюванні прийнято ряд припущень. По-перше, **стаціонарність між піками**: у проміжках часу поза періодами пікового навантаження потік замовлень вважається стаціонарним (інтенсивність надходження замовлень постійна). По-друге, **незалежність замовлень**: вважається, що часи надходження окремих замовлень є незалежними випадковими величинами, а сам потік можна наближено моделювати як пуассонівський. Це означає відсутність групових або пов’язаних замовлень – кожне замовлення генерується незалежно від інших. Також передбачається, що всі замовлення однакові за пріоритетом і вимоги до ресурсів, тобто система обробляє їх за чергою надходження (якщо не зазначено іншого).

Додатково, для спрощення моделювання **ігноруються можливі конфлікти руху AGV**: вважається, що AGV не блокують один одного на шляхах (або такі ситуації вирішуються ідеально), тож час поїздки залежить лише від відстані та швидкості. AGV пересуваються зі сталою середньою швидкістю без врахування прискорень чи уповільнень. Якщо моделюється відбір товару людиною, вважається, що час відбору є додатковою затримкою, але взаємодія між працівниками та AGV не детальна. Таким чином, модель фокусується на головних процесах (черги замовлень,

відбір, доставка) і абстрагується від другорядних деталей, забезпечуючи баланс між реалістичністю та складністю.

2.2 Вимоги

Функціональні вимоги. Система моделювання має підтримувати декілька основних режимів роботи. По-перше, **Single Run** – виконання одиночної симуляції для заданого сценарію. Користувач задає конфігурацію сценарію (кількість AGV, інтенсивність замовлень, тощо), після чого модель програє хід подій і формує результати. По-друге, **Experiments** – пакетний запуск серії симуляцій для різних конфігурацій з метою експериментального дослідження. Цей режим автоматично варіює задані фактори (наприклад, кількість роботів, політику диспетчеризації) і збирає зведену статистику для подальшого аналізу. Також система повинна генерувати **звіти** – зручне представлення результатів симуляції (графіки, таблиці ключових показників). Передбачено можливість **збереження/завантаження (Save/Load)** конфігурації сценарію та результатів: користувач може зберегти налаштування моделі чи отримані дані для повторного використання або аналізу. Наприклад, результати серії експериментів зберігаються у файл формату CSV або генеруються у вигляді HTML-звіту для перегляду в браузері.

Нефункціональні вимоги. Модель повинна забезпечувати **відтворюваність** результатів – при фіксованому початковому стані та значенні зерна випадкових чисел кілька запусків дають однаковий результат. Це досягається можливістю встановлення фіксованого random_seed для всіх стохастичних процесів (інтервали між замовленнями, поломки тощо). **Валідність** моделі є критичною: логіка симуляції має адекватно відображати реальну систему. Для цього результати моделювання зіставляються з аналітичними оцінками або емпіричними даними (наприклад, перевіряється, що при невеликому навантаженні час очікування близький до нуля, або що при дуже великому навантаженні система демонструє насичення черги). Модель повинна підтримувати достатню **швидкодію**, щоб забезпечити виконання

сотень експериментів у розумний час. Для цього критичні обчислення (напр. призначення завдань роботам) оптимізовані за допомогою ефективних алгоритмів (зокрема, використано OR-Tools для задачі призначення).

KPI та сценарії. Система розраховує низку **ключових показників ефективності (KPI)** для оцінки роботи складу. Серед них – середній час циклу замовлення (від надходження до виконання), середній час очікування в черзі, пропускна здатність (замовлень на годину), завантаженість ресурсів (відсоток часу, коли AGV зайняті, та ін.) тощо. Користувач визначає **сценарій** через конфігурацію: кількість і характеристики AGV (швидкість, надійність), параметри складу (топологія графа, стратегія зберігання товарів), режим відбору товарів (напр. людиною або автоматизований), профіль попиту (базова інтенсивність та пікові коефіцієнти), тривалість симуляції, тощо. Вимоги до сценаріїв включають гнучкість налаштування всіх цих параметрів і можливість легко додавати нові сценарії. Кожен сценарій може містити також налаштування політики диспетчеризації (правила призначення замовлень роботам) і опції збору даних (частота телеметрії, прогрів моделі та ін.).

2.3 Архітектура рішення (логічна й модульна)

Архітектура програмного рішення моделювання реалізована модульно, щоб спростити підтримку та розширення системи. Основні компоненти та їх взаємозв’язки показані на **рис. 2.1**. Користувацький інтерфейс на базі **Streamlit (UI)** забезпечує взаємодію з користувачем: через нього можна задавати параметри сценарію, запускати одиночну симуляцію або серію експериментів. Якщо користувач обирає Single Run, UI передає конфігурацію сценарію у модуль **run_advanced**, який відповідає за запуск симуляції. У випадку пакетного запуску (**Experiments**), UI надсилає налаштування серії у модуль **experiments/runner**, який в циклі генерує різні конфігурації і викликає run_advanced для кожної з них.

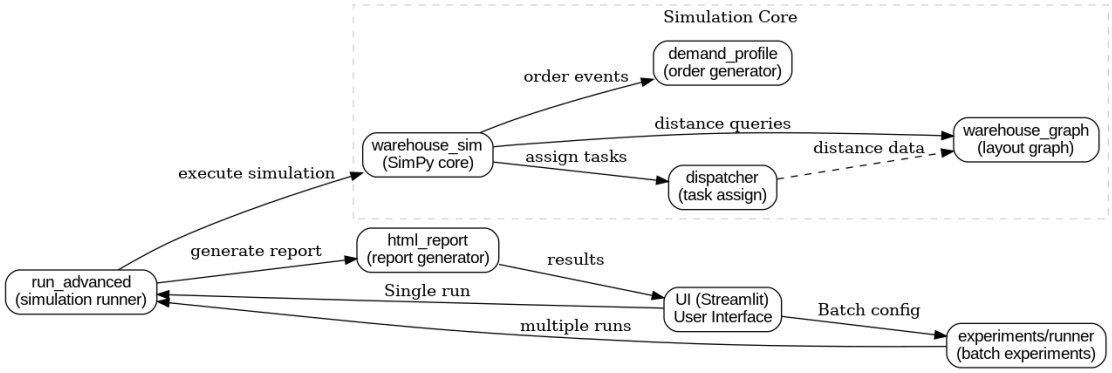

Рисунок 2.1 – Логічна схема компонентів системи моделювання складу і потоки даних між ними.

UI (користувацький інтерфейс на Streamlit) дозволяє запускати одиночний сценарій через виклик run_advanced або серію експериментів через experiments/runner. run_advanced – основний модуль запуску симуляції: він ініціалізує середовище SimPy та всі необхідні компоненти моделі, запускає симуляцію і по завершенні передає результати до модуля формування звіту html_report. Пакетний режим experiments/runner автоматично виконує кілька запусків run_advanced з різними параметрами і може акумулювати зведену статистику.

Simulation Core. В центрі архітектури – **ядро симуляції** (warehouse_sim), реалізоване із використанням бібліотеки дискретної симуляції SimPy. Воно відповідає за моделювання подій у часі: надходження замовлень, рух та дії AGV, обслуговування черг тощо. Модуль warehouse_sim використовує допоміжні компоненти для виконання спеціалізованих завдань. Зокрема, блок **demand_profile** генерує події надходження замовлень у відповідності до заданого профілю попиту (використовуючи параметри інтенсивності та розкладу піків, див. розд. 2.8). Нові замовлення додаються у систему (в чергу) як події SimPy з інтервалами, що відповідають статистичному розподілу міжприходів.

Для прийняття рішень про призначення замовлень роботам warehouse_sim звертається до модуля **dispatcher**. **Диспетчер** реалізує політику розподілу завдань між наявними AGV. Зокрема, він збирає інформацію про поточні вільні AGV та

невиконані замовлення і викликає оптимізаційний алгоритм (через OR-Tools) для визначення найкращого парування «AGV-замовлення». Отримавши призначення, диспетчер видає команди конкретним роботам на виконання доставок.

Компонент **warehouse_graph** містить модель складу у вигляді графа. Він надає методи для визначення найкоротшого шляху та часу пересування між будь-якими двома точками складу для заданої швидкості AGV. warehouse_sim користується цим компонентом, щоб обчислити час руху робота до місця відбору та до зони доставки. Диспетчер також може звертатися до warehouse_graph для оцінки витрат (відстаней), необхідних при побудові матриці призначення.

По завершенні симуляції, модуль run_advanced отримує деталізовані результати – **телеметрію** (часові ряди стану системи) та зведені показники KPI – і передає їх у компонент **html_report**. Той, у свою чергу, формує підсумковий звіт (напр. у форматі HTML із графіками і таблицями) та передає його на UI для відображення користувачу. Таким чином, архітектура забезпечує розподіл відповідальності між компонентами: генерація навантаження, диспетчеризація, підрахунок метрик і представлення результатів реалізовані окремо, що підвищує гнучкість системи.

2.4 UML-моделі

Для кращого розуміння вимог і поведінки системи побудовано кілька діаграм UML, що відображають різні аспекти функціонування моделі.

Діаграма варіантів використання. На рис. 2.2 зображено основні сценарії взаємодії користувача з системою. Актор **«Користувач»** ініціює запуск симуляції у одному з двох режимів: **«Запуск одиночної симуляції»** або **«Запуск серії експериментів»**. Перший варіант передбачає, що користувач налаштовує один сценарій та отримує результат його моделювання. Другий варіант – користувач задає діапазон параметрів або набір сценаріїв, і система автоматично виконує кілька симуляцій (експериментів) для порівняння результатів. Після виконання симуляції

користувач також може **«Отримати звіт»** – цей варіант використання реалізується автоматично після завершення моделювання, коли згенерований звіт стає доступним для перегляду або завантаження.

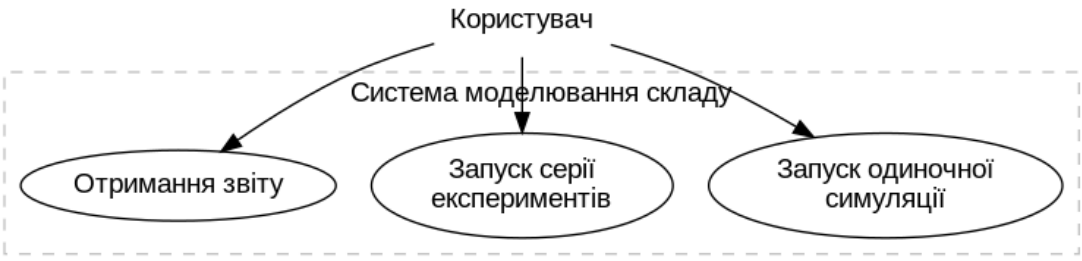

Рисунок 2.2 – Діаграма варіантів використання системи.

Діаграма активності. На рис. 2.3 показано типовий цикл життя одного замовлення у моделі, від його надходження до виконання. Нове замовлення спочатку **надходить і стає в чергу** очікування обробки. Система перевіряє, чи є зараз **вільний AGV** для його обслуговування. Якщо ні, замовлення чекає (залишається в черзі) до моменту, поки якийсь AGV не звільниться. Як тільки знаходиться вільний AGV, відбувається **призначення замовлення цьому AGV** (диспетчер видає команду на виконання). Далі AGV починає виконувати завдання: **прямує до пункту відбору товару**, де **забирає товар**, після чого **доставляє товар до пакувальної зони** (зони відправки). Коли доставка завершена, замовлення позначається як **виконане**, і процес для даного замовлення завершується. AGV при цьому звільняється і стає доступним для нового завдання. В діаграмі також показано цикл очікування: поки замовлення знаходиться в черзі, система періодично перевіряє, чи не з’явився вільний робот (подія «AGV звільнився»). Лише коли умова наявності вільного AGV стане істинною, замовлення переходить до стадії призначення і далі по процесу.

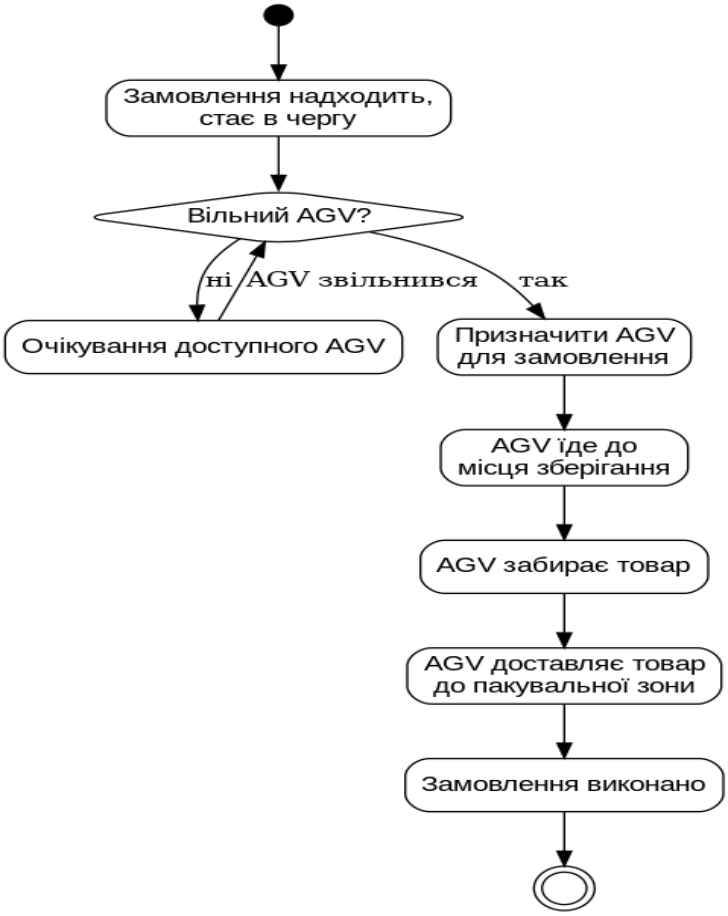

Рисунок 2.3 – Діаграма активності, що моделює обробку одного замовлення.

Діаграма послідовності. Рис. 2.4 ілюструє обмін повідомленнями між об’єктами системи під час **призначення завдання AGV** (диспетчеризації) при надходженні нового замовлення. Коли **черга замовлень** отримує нове замовлення, вона надсилає повідомлення диспетчеру про необхідність призначення (**«нове замовлення»** на діаграмі). Об’єкт **диспетчер** у відповідь ініціює процес призначення: він збирає дані про всі вільні AGV та невиконані замовлення і формує задачу оптимізаційної розстановки. Для цього диспетчер звертається до **OR-Tools solver** із запитом **«побудувати матрицю витрат і здійснити призначення»** – фактично передає матрицю витрат, де рядки відповідають вільним роботам, стовпці – відкритим замовленням, а елементи містять час (витрати) на виконання тим чи іншим роботом того чи іншого завдання. Розв’язувач OR-Tools виконує обчислення і

повертає **«результат призначення»** – оптимальний розподіл пар «AGV-замовлення». Отримавши результати, диспетчер надсилає відповідному **AGV команду на виконання** конкретного замовлення. Після цього AGV починає виконувати доставку. Діаграма відображає синхронний характер взаємодії: диспетчер очікує на відповідь від OR-Tools перед тим, як віддати команду роботам. (Для простоти у діаграмі не показано підтвердження від AGV, хоча в реалізації робот може підтвердити отримання завдання.)

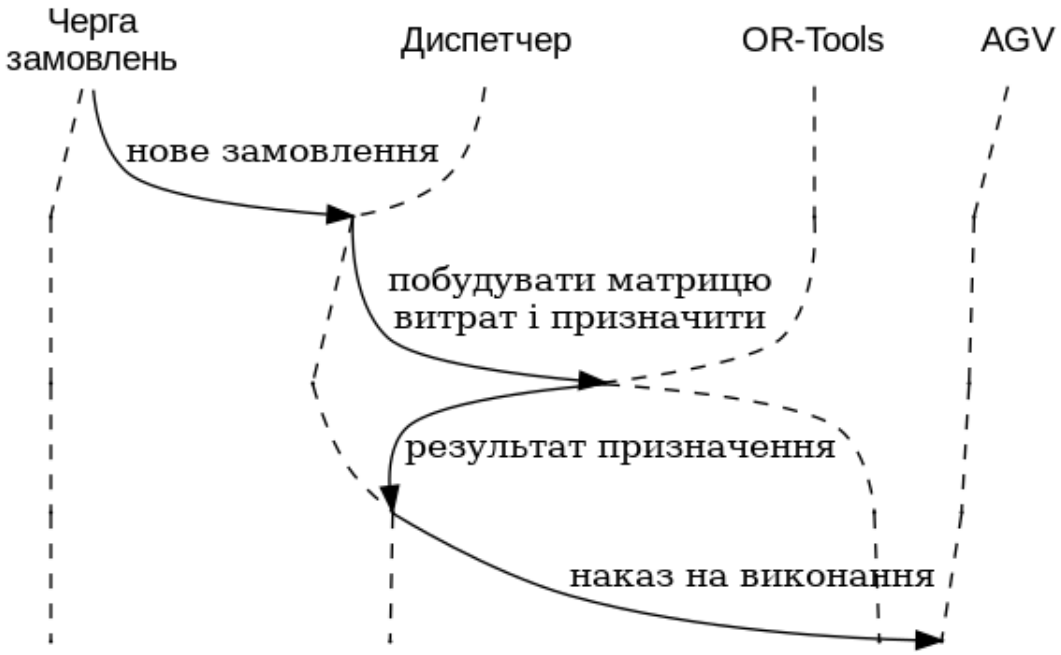

Рисунок 2.4 – Діаграма послідовності для процесу призначення замовлення AGV.

Діаграма станів. На рис. 2.5 подано спрощену діаграму станів одного **AGV** у моделі. У початковому стані робот **«Вільний»** – не виконує жодного завдання і очікує призначення. Коли диспетчер призначає роботу нове замовлення, AGV переходить у стан **«Прямує до пункту відбору»** – рухається по складу до місця зберігання необхідного товару. Досягнувши пункту відбору (стан **«Виконує відбір»**, тобто відбувається завантаження товару на робот або на носій), AGV потім перемикається у стан **«Доставляє вантаж»** – транспортує товар до виходу

(пакувальної зони). Після успішної доставки робот повертається у стан **«Вільний»**, сигналізуючи диспетчеру про готовність до нового завдання.

На діаграмі також відображено обробку **поломок AGV**. У будь-якому активному стані (в русі чи при роботі) може статися збої – AGV переходить у стан **«Не в строю (ремонт)»**. Передбачається, що несправність усувається за деякий час (параметр MTTR), після чого робот **«відновлено»** і він повертається до стану **«Вільний»**. (В даній моделі після поломки поточне завдання скасовується та буде переназначене іншому або тому ж AGV заново.) Таким чином, діаграма станів показує як основний цикл роботи AGV, так і відхилення, спричинені можливими відмовами техніки.

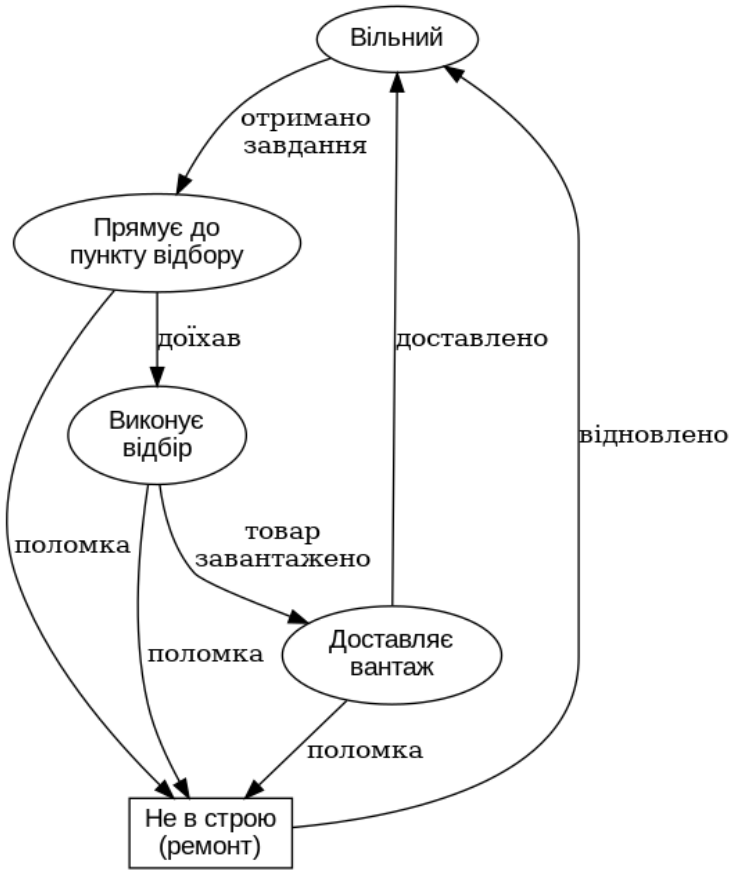

Рисунок 2.5 – Діаграма станів для одного AGV.

2.5 Мережа черг та подійна модель

Процеси на складі можна уявити як **мережу черг** масового обслуговування, що послідовно опрацьовують замовлення

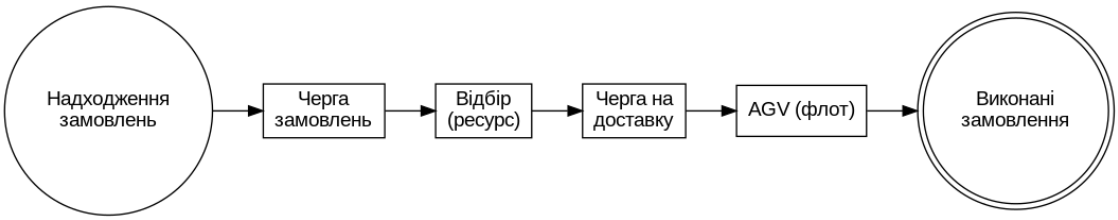

Рисунок 2.6 – Покрокова модель опрацювання замовлення.

Нові замовлення надходять у систему з певною інтенсивністю (відповідає джерелу заявок на схемі – «надходження замовлень»). Якщо на момент приходу замовлення ресурси, необхідні для його негайної обробки, зайняті – замовлення потрапляє в **чергу очікування**. Першим етапом обробки є **відбір товару (пікінг)** – на схемі це зображено як окремий ресурс. Залежно від сценарію, відбір може виконуватися людиною або автоматизованою системою: наприклад, працівник відбирає товар зі стелажа і передає його для подальшої доставки, або ж сам робот забирає товар (якщо це роботизований склад). У моделі даний етап налаштовується через параметр picking_mode: при **manual picking** замовлення спочатку чекає на доступність працівника/стелажа, виконується відбір, і лише потім передається в чергу на доставку AGV; при **automated picking** роль окремої черги відбору мінімізована, і процес відбору інтегрований з роботом.

Наступним етапом є **доставка замовлення за допомогою AGV**. Замовлення, готове до транспортування, стає в чергу на призначення роботу (на схемі – «черга на доставку»). Якщо при надходженні такого завдання є вільний AGV, черга може бути порожньою і завдання відразу призначається. В іншому разі замовлення чекає у черзі до звільнення першого доступного робота. Блок **AGV (флот)** на схемі виконує роль багатоканальної системи обслуговування: одночасно може працювати кілька AGV (кількість каналів дорівнює числу роботів). Кожен вільний AGV бере з черги

наступне замовлення і здійснює його доставку до виходу. Після завершення доставка замовлення виходить із системи («виконані замовлення»).

Подійна модель. Описана мережа черг реалізована як дискретно- подійна симуляція. Події включають: прибуття нового замовлення (генерує подію надходження та поміщення в чергу); початок обслуговування замовлення (коли ресурс стає доступним – працівник або AGV – замовлення виймається з черги і починає обслуговуватися); завершення обслуговування (після закінчення відбору або доставки товару). Диспетчеризація AGV вписана в цю модель подій – подія звільнення робота або прибуття нового завдання ініціює запуск алгоритму призначення (розд. 2.7), результатом якого є подія початку виконання завдання конкретним AGV.

2.6 Модель складу як граф

Топологія складу зображується у програмі у вигляді **графа вузлів і ребер**. Кожен вузол графа відповідає ключовій локації на складі: точці завантаження/вивантаження, перехрестю проходів, сектору зі стелажами тощо. Ребра графа представляють доступні шляхи пересування AGV між вузлами і характеризуються довжиною (відстанню). На рис. 2.6 подано фрагмент умовного графа складу: вузол **«Вихід»** (зона видачі замовлень) сполучений з основним коридором **«Перехрестя 1»** (відстань 10 м); від цього коридору відходять шлях до секції **«Склад 1»** (8 м) та далі до **«Склад 2»** через інший вузол **«Перехрестя 2»** (5 м + 7 м)

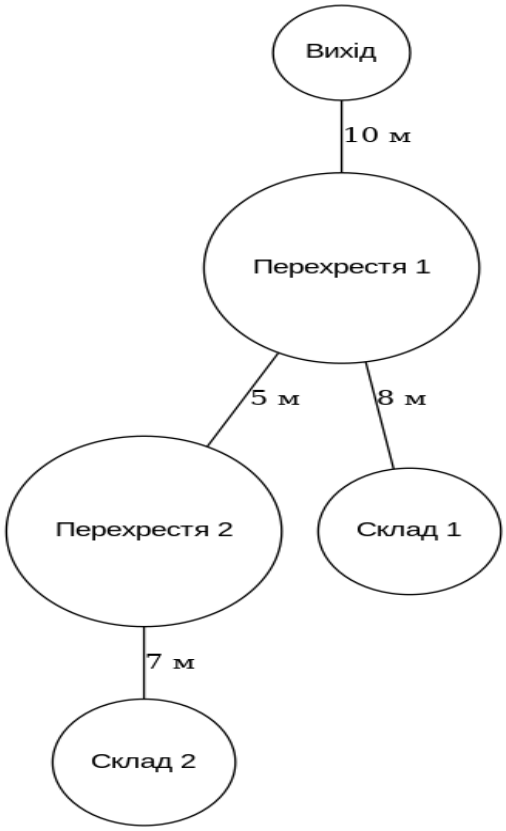

Рисунок 2.7 – Граф вузлів та ребер.

Таким чином, шлях від виходу до позиції «Склад 2» пролягає через два проміжні вузли з загальною довжиною 10+5+7=22 м.

Для оцінки часу переміщення робота між будь-якими двома точками використовується функція shortest_time_seconds(G, src, dst, agv_speed). Вона знаходить найкоротший шлях у графі G між вузлом-джерелом src і вузлом-призначення dst (наприклад, алгоритмом Дейкстри) та обчислює час як суму довжин ребер на цьому шляху, поділену на швидкість AGV. Формально, якщо $d(src,dst)$ – найкоротша відстань, а v – швидкість руху робота, то **час поїздки** оцінюється як:

$$T_{drive}(src, dst) = \frac{d(src, dst)}{v}.$$

Наприклад, для шляху довжиною 22 м при швидкості $v=1\,\text{м/с}$ час складає приблизно 22 с. Ця модель припускає, що роботу не доводиться зупинятися

чи сповільнюватися через інші роботи або перешкоди (як зазначалося в припущеннях, конфлікти руху ігноруються). Таким чином, AGV завжди може обрати найкоротший маршрут і рухатися зі сталою середньою швидкістю. Якщо у сценарії важливо врахувати затримки на поворотах чи перетинах, їх можна приблизно включити додаючи фіксовану затримку на кожен вузол або вводячи знижену ефективну швидкість. Але базовий сценарій моделі передбачає ідеальні умови руху для спрощення аналізу.

Описаний граф використовується як при обчисленні часу руху (для моделювання тривалості подій «їде до місця відбору» і «доставляє вантаж» у SimPy), так і для оцінки витрат при диспетчеризації. Диспетчер (розд. 2.7) при побудові матриці витрат використовує відстань від поточної позиції кожного вільного AGV до місця знаходження замовлення (стелажа) як одну зі складових часу виконання завдання цим роботом.

2.7 Призначення завдань AGV

Алгоритм диспетчеризації визначає, який робот візьме чергове замовлення на доставку, і реалізований у модулі **dispatcher**. Основне завдання – розв’язати задачу призначення оптимальним чином, особливо коли одночасно з’являється кілька замовлень або звільняється кілька AGV. В нашій постановці це зводиться до класичної задачі призначення (assignment problem): є множина O невиконаних замовлень, що очікують у черзі, та множина R вільних AGV; потрібно знайти відповідність між O і R, мінімізуючи загальні «витрати» виконання – зазвичай, сумарний час доставки всіх призначених замовлень.

Як **функція витрат** береться орієнтовний час виконання замовлення конкретним роботом. Він може складатися з двох компонент: часу проїзду від поточного місця знаходження AGV до місця відбору замовлення + часу доставки товару до виходу. Перший компонент система отримує через warehouse_graph (як описано в розд. 2.6), другий – аналогічно (від стелажа до виходу). У простішому

випадку, якщо вважати, що після прибуття до стелажа товар забирається миттєво і робот одразу їде до виходу, можна агрегувати ці два відрізки в один шлях від поточної позиції AGV до виходу через точку стелажа. Але в реалізації для гнучкості обчислюється окремо шлях до стелажа і від стелажа до виходу; сумою виходить повний час виконання. Так для кожної пари (робот *r in R*, замовлення *o in O*) визначається вартість *cost(r,o)*. Якщо замовлень більше, ніж роботів, частина замовлень залишиться непризначеною (чекатиме), а якщо роботів більше – деякі роботи залишаться без завдання (залишаться в простої).

Задача призначення розв'язується при настанні подій, що можуть змінити оптимальний розподіл: коли **нове замовлення надходить** (з'явився додатковий «попит» на ресурс) або **AGV звільняється** після виконання завдання (з'явилася додаткова одиниця ресурсу). У ці моменти диспетчер збирає актуальні множини *O* і *R* і запускає процедуру оптимізації. В нашій системі використано бібліотеку **Google OR-Tools**, яка містить ефективний реалізований алгоритм призначення на основі угорського алгоритму (*O(n^3)* у гіршому випадку, де *n=max(|O|,|R|))*[1]. Це дозволяє перебирати варіанти призначення досить швидко навіть при десятках роботів і завдань одночасно.

Нижче подано спрощований **псевдокод** роботи диспетчера при настанні відповідної події:

```
upon event (new_order_arrived or agv_became_free):
  O = list of pending orders (waiting in queue)
  R = list of free AGVs (idle robots)
  if O is not empty and R is not empty:
    let m = |R|, n = |O|
    cost_matrix = matrix of size (m x n)
    for i in range(m):       # for each AGV
      for j in range(n):    # for each order
        cost_matrix[i][j] = estimated_time_to_deliver(order_j by robot_i)
```

```
assignment = ORTools.SolveAssignment(cost_matrix)
for each (i,j) in assignment.pairs:
    assign order_j to robot_i   # dispatch AGV i to handle order j
```

Якщо вхідні множини O і R порожні (немає кого призначати або нікому виконувати), алгоритм нічого не робить. У разі, коли завдань більше, ніж роботів, OR-Tools знайде оптимальне призначення для підмножини замовлень розміром $|R|$ (решта продовжать чекати). Якщо роботів більше, ніж завдань, частина роботів залишиться без роботи; за потреби це теж можна трактувати як результат призначення (надлишкові роботи ігноруються).

Політика диспетчеризації. У даній моделі прийнята політика негайного призначення (immediate dispatch): щойно з'являється можливість призначити завдання, воно призначається. Альтернативою могла б бути періодична або пакетна диспетчеризація – коли система накопичує, скажімо, кілька нових замовлень і призначає їх раз на певний інтервал часу всі разом. Проте для спрощення реалізації і мінімізації часу очікування використовується саме подієвий підхід: кожне нове замовлення або звільнення робота тригерить алгоритм призначення. Це гарантує, що вільні ресурси не простоюють, а замовлення не очікують довше необхідного. Якщо ж потрібно обмежити «поспішність» призначень (щоб, наприклад, за дуже короткі інтервали не викликати алгоритм надто часто), в dispatcher передбачено таймер – мінімальний крок часу між послідовними перерахунками призначень. Також можливе застосування жадібної політики як спрощення: призначати нове замовлення найближчому вільному роботу без глобальної оптимізації. Такий режим можна включити для порівняння в експериментах (див. розд. 2.11). Очікується, що оптимальний алгоритм (OR-Tools) дасть кращі результати в метриках часу доставки, ніж жадібний, особливо при накопиченні черги, що й буде перевірено експериментально.

2.8 Механіка попиту і піковий профіль

Надходження замовлень у моделі задається стохастичним процесом з змінною інтенсивністю. Параметр **mean_interarrival_s** визначає середній інтервал між прибуттям двох замовлень у спокійний період (поза піками) – фактично це обернена величина до базової інтенсивності *lambda* (наприклад, mean_interarrival_s = 30 с відповідає приблизно lambda = 0.033$ замовл./с або 2 замовлення/хв).

Для моделювання коливань навантаження протягом доби або зміни введено поняття **пікового профілю навантаження**. Він задається структурою **peak_demand_schedule** – упорядкованим списком інтервалів, кожен з яких містить час початку піку (у хвилинах від початку симуляції) та множник інтенсивності. Множник означає, у скільки разів частота замовлень зростає відносно базової після настання даного моменту. Кожен наступний запис у списку змінює інтенсивність потоку на відповідний коефіцієнт. Наприклад, профіль може бути заданий як:

peak_demand_schedule = [(0, 1.0), (60, 3.0), (120, 0.5)]

Це означає, що з самого початку (0 хв) інтенсивність становить 1.0 times lambda (базовий рівень), починаючи з 60-ї хвилини зростає до 3.0 times lambda (пік, утричі вище базового), а з 120-ї хв падає до 0.5 times lambda (вдвічі нижче базового – період «долини»).

Рисунок 2.8 – Графік піковаго профілю навантаження.

До 1-ї години моделюється стабільний потік ~1 замовлення/умовний інтервал; в годину пік (1–2 година) частота надходження потроюється; далі на 3-й годині настає затишшя із вполовину нижчою від базової інтенсивністю.

Математично зміну інтенсивності можна інтерпретувати як **неоднорідний пуассонівський процес (NHPP)**, де моментна інтенсивність lambda(t) є функцією часу згідно з розкладом. У моделі реалізація цього процесу відбувається через динамічне планування інтервалів між замовленнями: на початку кожного інтервалу перевіряється, який множник зараз діє, і згідно з ним генерується наступний час прибуття. Зокрема, якщо поточний множник *m*, то інтервал *Delta t* до наступного замовлення вибирається випадково за експоненційним розподілом зі середнім *frac{text{mean_interarrival_s}}{m}*. Коли симуляційний час досягає точки зміни інтенсивності, значення m оновлюється і наступні інтервали генеруються вже з новим середнім. Така схема еквівалентна класичному thinning-алгоритму для NHPP [2] і забезпечує потрібну статистику прибуття замовлень.

Піковий профіль дозволяє моделювати сценарії, коли навантаження на склад нерівномірне – наприклад, вдень замовлень більше, а вночі менше; або під час акції/розпродажу бачимо сплеск замовлень. Це є важливим фактором, який впливає на продуктивність: в розділі експериментів 2.9 буде досліджено, як система справляється з піками і чи вистачає ресурсів, щоб уникнути великих черг у ці періоди.

2.9 План експериментів

Для оцінки впливу різних факторів на продуктивність системи підготовлено план експериментальних симуляцій. В рамках цього плану варіюються такі **фактори**:

– **Політика диспетчеризації (dispatch)**: порівнюються, наприклад, оптимальна (на основі OR-Tools) та жадібна (найближчий вільний AGV бере

замовлення) політики призначення. Це покаже, наскільки виграється час доставки від глобальної оптимізації.

– **Стратегія зберігання (storage)**: розподіл товарів по складу. Розглядаються варіанти Random (товари розміщені випадково) проти ABC (popularity-based), за якої найбільш ходові товари ближче до виходу. Очікується, що оптимізована стратегія зменшить середній шлях доставки.

– **Режим відбору (picking_mode)**: manual vs automated. У manual – додається час відбору людиною і можливо черга на відбір, в automated – роботи самі забирають товари без додаткової черги. Це впливає на час циклу замовлення і потребу в людських ресурсах.

– **Кількість AGV (agvs)**: різні значення, напр. 3, 5, 7 роботів. Цей фактор безпосередньо визначає пропускну здатність на етапі доставки; при інших рівних, більше роботів має зменшити чергу і час очікування, але зменшення може мати межу (якщо інші етапи стають лімітуючими).

– **Надійність AGV (MTBF, MTTR)**: вводяться сценарії з різною середньою напрацювання на відмову, напр. без відмов (MTBF = ∞), середня надійність (MTBF = 1000 хв), низька надійність (MTBF = 300 хв). Для кожного з них можна варіювати час відновлення – напр. MTTR = 0 (миттєвий перезапуск) vs MTTR = 60 хв (тривалий ремонт). Ці фактори впливають на доступність роботів і можуть суттєво збільшувати черги при низькій надійності.

– **Профіль навантаження (peak profile)**: тестуються різні профілі – рівномірний (без піків), помірний пік (наприклад, множник 2 в години пік), екстремальний пік (множник 3-4). Це демонструє стійкість системи до навантажувальних періодів.

Дизайн експерименту. Планується проведення повного факторного експерименту з усіма переліченими факторами. У таблиці 2.5 наведено варіанти рівнів для кожного фактора. Загальна кількість сценаріїв дорівнює добутку кількості рівнів (при 2 рівнях для більшості та 3 рівнях для кількох – отримуємо десятки

варіантів). Кожен сценарій буде пронумеровано, і для надійності виконано кілька прогонів з різними випадковими посівами, аби усереднити стохастичні коливання.

Таблиця 2.5

Варіювання факторів у експериментальному дизайні

Фактор	**Рівні (варіанти)**
Dispatch policy	Greedy (жадібний), Optimal (OR-Tools)
Storage strategy	Random (випадкова), ABC (популярність)
Picking mode	Manual (ручний відбір), Automated (роботиз.)
Number of AGVs	3, 5, 7 (шт)
MTBF (надійність)	∞ (без відмов), 1000 хв, 300 хв
MTTR (ремонт)	0 хв, 60 хв
Peak load profile	None (плаский), Moderate (×2), High (×3)

Метрики відповіді. Для кожного сценарію збираються KPI, описані в розд. 2.9: середній час циклу, середній час очікування, максимальна довжина черги, утилізація роботів, відсоток невиконаних замовлень до кінця симуляції тощо. Аналіз впливу факторів буде проводитись методами планування експериментів: побудова таблиці середніх значень метрик для кожного рівня фактора (з усередненням за іншими факторами), розрахунок ефектів і їх значущості. Очікується, що найбільший вплив на час виконання замовлень матимуть **кількість AGV** та **інтенсивність попиту**, оскільки вони безпосередньо визначають відношення навантаження до пропускної спроможності. Значний ефект також прогнозується від **диспетчерської політики** – оптимальна повинна зменшити зайві пробіги і час очікування в порівнянні з жадібною. Надійність AGV (MTBF/MTTR) може позначитися на метриках лише у сценаріях з високим навантаженням, коли відмови призводять до браку роботів; в легших режимах навіть часті відмови можуть не створити черг (якщо роботи встигають відновитися раніше, ніж накопичиться черга). **Режим**

відбору та **стратегія зберігання** ймовірно матимуть менш виражений вплив, але теж помітний: ручний відбір додає сталу затримку до кожного замовлення, а погана організація зберігання збільшує середню відстань доставки – ці фактори здатні збільшити час циклу, хоча й не впливають на пропускну здатність докорінно.

Для наочності результати планується представити у вигляді графіків ефектів.

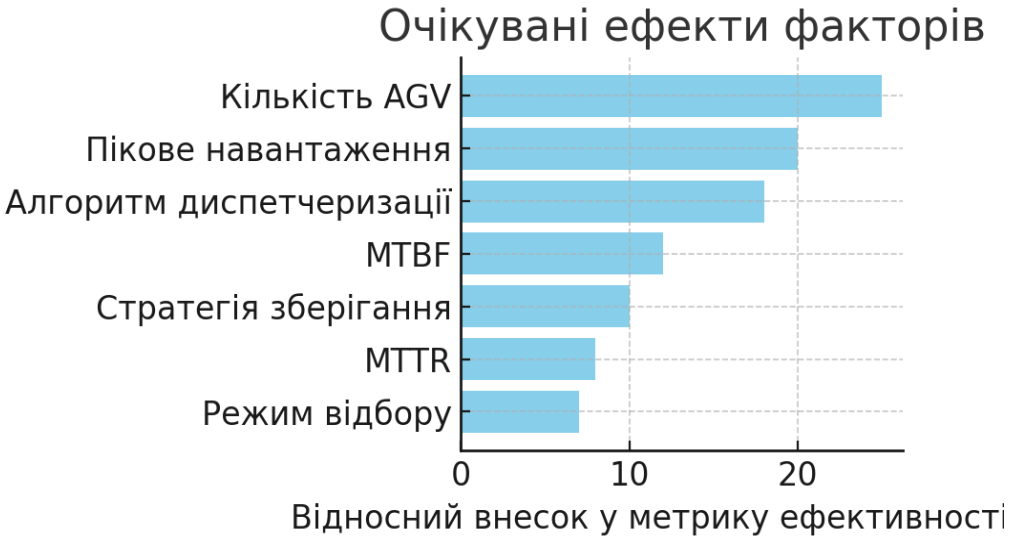

Рисунок 2.9 – Карта очікуваних ефектів.

Тут по горизонталі відкладено оцінка відносного впливу кожного фактора на цільову метрику (скажімо, на середній час циклу замовлення) у відсотках. Видно, що, згідно очікувань, **кількість AGV** та **пікове навантаження** (інтенсивність) займають найбільшу долю у варіації результату – разом понад 45%. На третьому місці – **алгоритм диспетчеризації**, який може пояснювати близько 18% зміни метрики. Фактори надійності (MTBF) та стратегія зберігання оцінюються в районі 10% кожен, а режим відбору і MTTR мають найменший внесок (біля 5-8%). Звичайно, точні значення буде отримано після обробки результатів симуляцій, але такий попередній аналіз допомагає сформулювати гіпотези. Зокрема, перевіримо, чи справді взаємодія між кількістю роботів і політикою призначення істотна (можливо, при великому надлишку роботів різниця між політиками згладжується), або як саме

впливає профіль пікового навантаження – лінійно чи має точку насичення, коли система перестає справлятися.

Зрештою, експериментальний план дозволить комплексно оцінити систему і дати рекомендації: скільки роботів достатньо для заданого характеру попиту, чи варто впроваджувати складні алгоритми диспетчеризації, яку стратегію зберігання обрати тощо. Результати аналізу будуть представлені у розділі 3 (Аналіз результатів).

РОЗДІЛ 3. ТЕХНОЛОГІЧНІ АСПЕКТИ

3.1 Загальний огляд реалізації

Симулятор автоматизованого складу реалізовано мовою Python із використанням низки сучасних бібліотек. Основою є дискретно-подійна симуляція на базі **SimPy**, що відповідає за часову модель та паралельне виконання процесів. Для моделювання топології складу та шляхів руху застосовується **NetworkX** (пошук найкоротших шляхів алгоритмом Дейкстри), а для оптимізації призначення завдань роботам – пакет **OR-Tools** від Google (розв'язання задачі призначення). Візуалізація та взаємодія з користувачем здійснюється через веб-інтерфейс на **Streamlit** з інтерактивними графіками (бібліотека Plotly). Додатково, для збору та аналізу даних симуляції використано pandas, а для генерації звіту – matplotlib (графіки конвертуються у зображення та вставляються в HTML через base64-кодування).

Архітектура програмного проєкту розбита на кілька модулів. У каталозі **src/** знаходиться вихідний код симулятора:

– **src/sim/** – ядро симуляції та допоміжні компоненти: warehouse_sim.py (основний модуль моделі складу), warehouse_graph.py (формування графа складу та функції маршрутизації), dispatcher.py (алгоритми призначення завдань), config.py (клас конфігурації SimConfig).

– **rc/report/** – генерація звітів: html_report.py формує HTML-звіт із результатів симуляції.

– **src/experiments/** – пакет для експериментальних запусків: runner.py для серійних прогонів і обробки результатів.

– На верхньому рівні – скрипти run_scenario.py (спрощений запуск одного сценарію) та run_advanced.py (запуск сценарію з використанням класу SimConfig).

– **ui/** – файли інтерфейсу користувача на Streamlit: streamlit_app.py (основний файл веб-додатку), а також підмодуль ui/components/ (наприклад, допоміжні компоненти інтерфейсу, якщо виділені окремо).

– **api/** – реалізація REST API на базі FastAPI: api/app.py (веб-сервіс для запуску симуляції через HTTP-запити, отримання параметрів тощо).

– **data/** – дані та приклади: наприклад, layout_example.csv із описом макету складу (список вузлів і зв’язків), orders_example.csv – приклад фіксованого набору замовлень.

– **tests/** – модульні тести: файли test_*.py з перевірками базової функціональності (наприклад, чи запускається симуляція з дефолтними налаштуваннями, чи отримуються KPI, чи працює диспетчер тощо).

– **outputs/** (або **runs/**) – результати виконання симуляцій. За домовленістю, кожен запуск створює підпапку (наприклад, з часовою міткою 2025-11-12_13-00-00), де містяться файли: params.json (конфігурація запуску), summary.csv та timeseries.csv (зведені результати та часові ряди), report.html (згенерований звіт), можливо log.txt (журнал роботи). Це полегшує порівняння: можна потім зібрати всі summary.csv з різних запусків для аналізу.

– **requirements.txt** – перелік залежностей (SimPy, pandas, NetworkX, OR-Tools, FastAPI, uvicorn, Streamlit, Plotly тощо) для встановлення необхідних пакетів.

– **README.md** – документація проєкту: опис призначення, інструкція зі встановлення та запуску. Наприклад, там наведено Quick start із командами для трьох режимів: запуск через CLI (python src/run_scenario.py), запуск UI (streamlit run ui/streamlit_app.py) або запуск API (uvicorn api.app:app).

Така модульна побудова робить симулятор гнучким і розширюваним. Різні користувачі можуть взаємодіяти з ним у зручний спосіб: аналітики – через пакетні експерименти та звіти, операційні менеджери – через інтерактивний веб-інтерфейс, розробники – через API або безпосередньо викликами класів. Це відповідає вимогам до MVP системи керування автоматизованим складом, демонструючи її технологічну спроможність і адаптивність.

3.2 Ядро симуляції (SimPy)

Ядро симуляції реалізовано у модулі **warehouse_sim.py** і відповідає за динаміку роботи складу – обробку замовлень, переміщення AGV, діяльність пікерів та збір ключових показників. Модель побудована за парадигмою дискретно-подієвої симуляції з використанням середовища SimPy. У процесі ініціалізації створюється середовище env = simpy.Environment() та паралельно запускається кілька процесів (генерація замовлень, диспетчер задач, моніторинг тощо). Також задаються необхідні ресурси. Зокрема, у моделі визначено:

– **AGV** – генерується список self.agvs заданої кількості AGV (параметр конфігурації agvs). Кожному AGV присвоюється індекс та початкове розташування (як правило, зона доку або станція) і початковий стан (усі AGV стартують вільними, без завдання).

– **Пікери** – моделюються як ресурс self.pick_resource = simpy.Resource(capacity=cfg.pickers). Це означає, що одночасно відбирати товар можуть задана кількість працівників (або автоматизованих станцій) – наприклад, якщо pickers = 2, паралельно можуть виконуватися два процеси відбору (2 паралельні канали ресурсу).

– **Черга замовлень** – у моделі зберігається структура (наприклад, список) self.pending_tasks, куди потрапляють замовлення, що очікують обробки. Нові замовлення додаються генератором замовлень, а диспетчер вибирає з цієї черги завдання для призначення AGV.

– **Замовлення** – кожне нове замовлення може представлятися об’єктом (наприклад, екземпляром класу Order) або словником із полями: час надходження, перелік необхідних SKU, кількість товарів тощо. При генерації замовленню надається унікальний ID, генерується список товарів (SKU) випадково або за заданим профілем попиту, і фіксується поточний час як час надходження.

– **Ремонтні бригади** – якщо враховуються відмови обладнання, використовується ресурс self.repair_crews = simpy.Resource(capacity=X) для

моделювання бригад техобслуговування (за замовчуванням X = 1). Цей ресурс обмежує кількість одночасних ремонтів: наприклад, якщо доступна лише одна бригада, одночасно може ремонтуватися тільки один AGV, інші чекатимуть у черзі (детальніше про моделювання відмов – див. розд. 4.2).

Основні процеси симуляції запускаються під час виклику WarehouseSim.run(). Серед них:

– **Генератор замовлень** – процес, що моделює надходження нових замовлень. Він працює у безкінечному циклі: генерує замовлення, додає його до черги pending_tasks і чекає інтервал до появи наступного. Інтервали моделюються випадково на основі заданої середньої інтенсивності (параметр mean_interarrival_s): зазвичай використовують експоненційний розподіл, щоб зімітувати пуассонівський потік замовлень. Так можна отримати випадкові часи прибуття заявок із заданим середнім. За потреби можна врахувати розклад – наприклад, зробити інтенсивність залежною від часу доби (пікові та непікові періоди, див. розд. 3.5). Генератор працює до завершення часу симуляції (наприклад, 240 хвилин) або поки кількість згенерованих замовлень не досягне наперед визначеного ліміту (в поточній реалізації ліміт не задано, тому генерує замовлення протягом усього часу моделювання).

– **Диспетчер задач** – процес, що постійно відслідковує чергу замовлень і доступні AGV та виконує призначення завдань (алгоритм описано в підрозд. 3.4). Фактично, диспетчер запускає процедуру assignment, коли є вільні роботи і невиконані задачі: він формує матрицю витрат і викликає оптимізаційний алгоритм. Після призначення кожному вільному AGV конкретного завдання диспетчер може чекати певний час (або подію надходження нового замовлення) перед наступним циклом розподілу.

– **Моніторинг/телеметрія** – додаткові процеси для збору статистики та логування під час симуляції. Наприклад, можна раз на певний інтервал записувати поточну довжину черги, кількість виконаних замовлень, завантаження ресурсів тощо

для побудови графіків. Ці процеси не впливають на саму динаміку, але накопичують дані для звітності.

Середовище SimPy запускається методом env.run(until=T), де T – задана тривалість модельного часу (наприклад, 14400 секунд = 240 хвилин). При досягненні часу T симуляція зупиняється – активні процеси перериваються, а накопичені результати стають доступними. Після виконання WarehouseSim.run() зібрані KPI, лог-дані та інші результати можна передати в модуль генерації звіту або проаналізувати.

3.3 Топологія складу (warehouse_graph.py)

Для точного моделювання переміщень AGV важливо задати топологію складу – план зон та шляхів – у вигляді графа. У системі використовується модуль **warehouse_graph.py**, який будує граф складу: вершини графа відповідають ключовим локаціям (станції приймання, пункти відбору, перехрестя, парковки), а ребра – можливим маршрутам між ними. Для побудови графа використовується бібліотека NetworkX. Граф може бути заданий кодом (наприклад, у вигляді списків вузлів та ребер) або завантажений із зовнішнього файлу (CSV зі списком вузлів і їхніх сусідів).

Кожне ребро графа має вагу, що відповідає часу руху по цьому сегменту (або відстані). На початку роботи симуляції граф завантажується або створюється – наприклад, з файлу **layout_example.csv** у папці data можна зчитати перелік вузлів та з’єднань між ними, збудувавши таким чином карту складу. Далі, при кожній потребі знайти найкоротший шлях (наприклад, при призначенні завдання чи для моделювання руху AGV) використовується алгоритм пошуку – NetworkX надає відповідні функції (Dijkstra або ін.). Результатом є список вузлів або ребер маршруту і сумарна вага (час/відстань).

У поточній моделі припускається, що всі основні шляхи закладені в графі й однаково доступні для роботів (двосторонній рух, якщо не зазначено односторонніх

сегментів). Якщо є вузькі проходи, де можливий лише односторонній рух, це можна відобразити через орієнтовані ребра. Також, за потреби, можна задавати різні графи для порожнього і завантаженого AGV (з різними швидкостями), але в наших експериментах використовується спрощення з єдиним значенням швидкості.

Модуль warehouse_graph.py надає функції для отримання відстані між двома точками (get_distance(node1, node2)) або найкоротшого шляху (get_path(node1, node2)). Ці можливості використовуються диспетчером при обчисленні матриці витрат (розд. 3.4) та самим AGV при русі (AGV запитує свій маршрут). Граф можна адаптувати або розширити: наприклад, якщо треба змоделювати перекриття проходу, відповідне ребро можна тимчасово видалити чи збільшити його вагу.

3.4 Призначення завдань (dispatcher.py)

Алгоритм диспетчеризації реалізовано у модулі **dispatcher.py**. Він відповідає на питання: який із вільних AGV яке завдання має отримати, щоб мінімізувати час виконання? Це – практична реалізація задачі призначення, описаної теоретично в розд. 1.6. У моделі диспетчер постійно або періодично перевіряє стан: чи є незайняті AGV та невиконані завдання (замовлення в черзі). Якщо так, він формує матрицю витрат c_{ij}: для кожного вільного робота i і кожного завдання j обчислюється оціночний час виконання – зазвичай час під'їзду робота до місця замовлення (оскільки решта операції для всіх роботів однакова). Цей час визначається за допомогою графа складу: береться поточна позиція *AGVi*, вузол призначення (найближчий до місця замовлення) і виконується пошук найкоротшого шляху, далі довжина маршруту ділиться на швидкість, отримуємо c.

Потім диспетчер викликає оптимізатор OR-Tools для розв'язання отриманої assignment-problem[1]. Задача призначення задається бібліотеці OR-Tools у вигляді матриці вартостей $[c_{ij}]$, і бібліотека знаходить оптимальне призначення (використовуючи угорський алгоритм або еквівалентний метод мінімального потоку). Результат – оптимальний набір пар (AGV, завдання), що мінімізує сумарний

час під’їзду роботів до своїх завдань. Кожен вільний AGV отримує щонайбільше одну задачу, невиконані завдання (якщо їх більше, ніж роботів) залишаються в черзі.

У нашій реалізації диспетчер виконує призначення практично безперервно: як тільки з’являються вільні AGV і невиконані замовлення, він негайно формує матрицю витрат. Значення витрат обчислюються з використанням функції пошуку найкоротшого шляху (модуль warehouse_graph). Отримавши результат оптимізації від OR-Tools, диспетчер присвоює кожному AGV його нове завдання та запускає відповідні процеси AGV у ядрі симуляції (робот починає рух до точки відбору). Потім диспетчер знову чекає появи нових задач або звільнення роботів, щоб повторити процедуру.

Таким чином, диспетчер виступає зв’язкою між чергою замовлень і парком роботів, намагаючись у кожен момент використати ресурси найефективніше. Застосування OR-Tools[1] дозволяє легко розв’язувати цю оптимізаційну підзадачу в режимі реального часу. Навіть для десятків роботів і задач обчислення триває частки секунди, тож алгоритм призначення не створює відчутних затримок у роботі моделі. Отримане "жадібно-оптимальне" призначення завдань мінімізує холості пробіги AGV і час очікування замовлень у черзі на кожному кроці.

3.5 Конфігурація симуляції (config.py)

Щоб зробити симулятор гнучким і придатним для різних сценаріїв, усі основні параметри винесені в окрему конфігураційну структуру – клас **SimConfig** (модуль config.py). У цій конфігурації задаються: кількість ресурсів (наприклад, agvs – кількість AGV, pickers – число пікерів), параметри продуктивності (швидкість AGV, середній час відбору одного товару пікером тощо), інтенсивність потоку замовлень (середній інтервал між замовленнями), політики управління (наприклад, режим відбору – хвилі чи безперервно), тривалість симуляції та інші налаштування.

Виділення SimConfig окремо від коду моделі дозволяє легко запускати різні сценарії без зміни програмної логіки. Достатньо створити чи відредагувати об’єкт

конфігурації. Наприклад, змінюючи SimConfig, можна прогнати сценарій "2 AGV, 1 пікер, хвильовий відбір" і одразу ж інший – "4 AGV, 2 пікери, безперервний відбір", не переписуючи код симулятора. У нашому проєкті передбачено збереження конфігурацій у JSON-файли, щоб користувач міг підготувати кілька сценаріїв і потім легко завантажувати їх для запуску (див. розд. 3.10). Такий підхід підвищує відтворюваність експериментів і зручність аналізу різних випадків.

3.6 Сценарії запуску (run_advanced.py)

Для керування запуском симуляції використовується модуль **run_advanced.py**, який дозволяє задавати сценарій на основі конфігурації, запускати його та отримувати результати. Цей модуль слугує зв’язковим між конфігурацією SimConfig і ядром моделі WarehouseSim.

Наприклад, у run_advanced можна написати код: завантажити SimConfig з JSON-файлу (або згенерувати програмно), створити об’єкт WarehouseSim з цим конфігураційним об’єктом, викликати sim.run(), а потім отримати з sim усі потрібні вихідні дані (метрики, логи). Цей модуль фактично інкапсулює послідовність дій з налаштування та виконання одного сценарію, що спрощує повторні запуски.

Модуль run_advanced корисний як для запуску симулятора з командного рядка (CLI), так і для викликів з UI чи інших програм. У Streamlit UI при натисканні кнопки "Start simulation" під капотом викликається саме функція запуску сценарію. Також run_advanced може зберегти отримані результати у файли (наприклад, записати summary.csv та timeseries.csv у папку **outputs/** для подальшого аналізу).

Таким чином, сценарний модуль відокремлює процедуру виконання симуляції від самої логіки моделі, що покращує структуру коду і полегшує підтримку програми.

3.7 Інтерфейс користувача (Streamlit UI)

Для взаємодії користувача з симулятором розроблено веб-інтерфейс на основі **Streamlit** (файл streamlit_app.py). Це дає змогу запускати сценарії без написання коду – через зручний веб-додаток. Користувач задає налаштування сценарію (кількість техніки, інтенсивність замовлень, варіанти політик тощо) за допомогою інтерактивних елементів (слайдерів, випадаючих меню) і запускає моделювання кнопкою.

Після запуску симуляції інтерфейс може в реальному часі відображати поточний стан (наприклад, лічильник виконаних замовлень, графік довжини черги тощо). По завершенні сценарію UI показує підсумкові результати у наочній формі. Зокрема, будуються графіки на базі Plotly: наприклад, графік динаміки довжини черги та кумулятивної кількості виконаних замовлень у часі, гістограми розподілу часу виконання замовлень, тощо. Крім того, виводяться основні KPI сценарію (throughput, середній lead time, % завантаження ресурсів) – їх можна показати у вигляді таблиці або окремих індикаторів.

Інтерфейс передбачає можливість експорту отриманих результатів. Є кнопка **"Download HTML Report"**, яка дозволяє завантажити сформований HTML-звіт із усіма графіками та параметрами сценарію (див. розд. 3.9). Це зручно для документування: користувач може після аналізу в інтерактивному режимі зберегти звіт для презентації чи звітності.

Таким чином, Streamlit UI робить інструмент доступним ширшому колу користувачів – людям, які не пишуть код, але хочуть проаналізувати роботу складу. Оператори чи аналітики можуть задавати сценарії, запускати симуляцію і одразу бачити результати, що значно прискорює цикл прийняття рішень.

3.8 Модуль експериментів (experiments/runner.py)

Для проведення серії експериментів із різними конфігураціями призначений модуль **experiments/runner.py**. Він дозволяє автоматизовано запускати симуляцію багаторазово з різними наборами параметрів і акумулювати результати.

Наприклад, якщо потрібно дослідити, який із двох алгоритмів диспетчеризації кращий, можна налаштувати experiments/runner так, щоб він виконав кілька запусків симуляції з першим алгоритмом і кілька – з другим (з відповідними конфігураціями). Потім модуль збере й усереднить результати кожної групи запусків, обчислить довірчі інтервали та згенерує порівняльний звіт. В UI можна відобразити ці результати у вигляді графіків: наприклад, стовпчикові діаграми з показниками для кожного сценарію чи "box plot" розподілу значень KPI.

Модуль experiments також підтримує багатофакторні експерименти. Можна задати список рівнів для кількох параметрів – наприклад, [2,4,6] AGV × [continuous, wave] політика – і він перебере всі комбінації, запустить симуляцію для кожної та збере результати. Це фактично реалізує описаний у розд. 1.8 план експерименту.

Підсумкова інформація, зібрана модулем experiments, може зберігатися у файл (наприклад, агрегована таблиця в CSV) або передаватися у UI для візуалізації. Цей модуль значно полегшує проведення обчислювальних досліджень: користувачу не потрібно вручну запускати десятки сценаріїв – достатньо описати діапазон параметрів, і система сама проведе всі прогони та видасть готові статистичні результати.

3.9 Генерація звітів (html_report.py)

Для представлення результатів симуляції у зручному, самодостатньому форматі передбачено модуль **html_report.py**, який формує HTML-звіт. HTML-звіт об’єднує в собі опис сценарію (його параметри) та ключові результати моделювання.

До звіту входять:

– **Параметри сценарію** – на початку звіту перераховуються основні налаштування (скільки AGV, скільки пікерів, яка середня інтенсивність замовлень, політика відбору тощо). Це важливо, щоб однозначно розуміти контекст результатів.

– **Таблиця KPI** – подаються значення основних показників: пропускна здатність, середній час циклу замовлення, рівень завантаження ресурсів, % вчасно виконаних замовлень тощо для даного сценарію.

– **Графіки** – включаються ключові графічні результати. Наприклад, графік динаміки довжини черги і кумулятивної кількості виконаних замовлень (з підписами типу "черга" і "виконано" різними кольорами), гістограма розподілу часу виконання замовлень, тощо. Графіки, побудовані за допомогою matplotlib, конвертуються в рядок base64 і вставляються прямо в HTML як зображення (тег <img src="..."> з отриманим рядком). Таким чином, HTML-файл містить усі графіки всередині себе і не залежить від зовнішніх файлів.

– **Пояснення** – за потреби, у кінці звіту можна додати текстові коментарі або висновки. У нашому модулі основний акцент на автоматичному генеруванні звіту, тому додаткових ручних висновків не передбачено – файл є самодостатнім носієм даних.

Згенерований звіт надає цілісний опис сценарію та його результатів. Його зручно пересилати чи додавати до документації. Особливо корисно, що він включає параметри сценарію – не виникає плутанини, про який саме запуск йдеться, якщо їх було декілька. Користувач, проаналізувавши результати в інтерфейсі, може одним кліком зберегти такий звіт (через кнопку в UI).

3.10 Структура проєкту і запуск проєкту

Як зазначалося, проєкт має чітку структуру директорій та файлів. Резюмуємо основне:

– **Каталог src/** – містить весь код: ядро симуляції (src/sim), модулі звітів (src/report), експериментів (src/experiments). Ключові файли: warehouse_sim.py, warehouse_graph.py, dispatcher.py, config.py (ядро); html_report.py (звіт); runner.py (експерименти); run_scenario.py і run_advanced.py (скрипти запуску).

– **Каталог ui/** – містить інтерфейс Streamlit: streamlit_app.py (основний файл) і, при наявності, ui/components/ для додаткових елементів інтерфейсу.

– **Каталог api/** – містить реалізацію FastAPI: файл api/app.py з описом веб-сервісу (ендпойнти для запуску симуляції і отримання результатів через HTTP).

– **Каталог data/** – вхідні дані: приклади файлів макету складу (layout CSV), файлів замовлень тощо.

– **Каталог outputs/** (runs/) – вихідні дані: результати виконання симуляцій, розкладені по папках запусків з файлами конфігурації, таблицями результатів, звітами.

– **requirements.txt** – перелік залежних пакетів для встановлення (SimPy, NetworkX, OR-Tools, pandas, Plotly, Streamlit, FastAPI тощо).

– **README.md** – містить огляд проєкту та інструкції. В ньому наведено приклади команд для запуску у трьох режимах:

Способи запуску симулятора:

1. Через консоль (CLI). Виконати команду, наприклад:

python src/run_scenario.py --minutes 240

Це виконає симуляцію тривалістю 240 хвилин з параметрами за замовчуванням (можна задати інші параметри через аргументи командного рядка, залежно від реалізації). По завершенні в консоль можуть бути виведені основні результати, а детальні – записані у файли (як описано вище). Цей режим дозволяє швидко запускати окремий сценарій з терміналу.

2. Через веб-інтерфейс Streamlit. Виконати команду:

streamlit run ui/streamlit_app.py

Це запустить локальний веб-сервер Streamlit. В консолі з’явиться адреса (типово **http://localhost:8501**), яку слід відкрити у браузері. Користувачу буде доступний інтерфейс, описаний у 3.7: можна налаштувати параметри сценарію, запустити симуляцію, переглянути результати та експортувати їх. Цей інтерактивний режим зручний для дослідження і демонстрації роботи моделі.

3. Як бекенд-сервіс (API). Виконати команду:

   uvicorn api.app:app --reload

Це стартує сервер FastAPI. В цьому режимі симулятор працює як веб-сервіс: можна відправити HTTP-запит із JSON-конфігурацією і отримати у відповіді JSON з результатами. Також FastAPI надає інтерактивну документацію (Swagger UI) за адресою **/docs**. Режим API не має графічного інтерфейсу, але дозволяє викликати симуляцію програмно, що відкриває можливості для розширень – наприклад, для оптимізаційних задач, де зовнішній код перебирає різні конфігурації, викликаючи симулятор як функцію.

Збереження та завантаження сценаріїв. Як зазначалося, система дозволяє зберегти конфігурацію сценарію у файл (JSON) через UI або CLI. Це забезпечує повторюваність: зберігши файл scenario_high_demand.json, користувач може пізніше завантажити його (UI: кнопка "Load scenario", CLI: аргумент --config scenario_high_demand.json) і виконати той самий сценарій ще раз. У JSON містяться всі поля SimConfig, тому навіть якщо у програмі додадуться нові параметри, збережені сценарії залишаться сумісними (невідомі поля просто ігноруються або приймаються рівними значенням за замовчуванням).

Отже, багатокомпонентна структура (ядро + UI + API + експерименти) робить симулятор модульним і розширюваним. Користувачі з різними потребами можуть взаємодіяти з ним зручним для них способом: аналітики – через експерименти і звіти, оператори – через інтерактивний інтерфейс, розробники – через API або прямі виклики класів у коді. Такий підхід дозволяє крок за кроком перетворити прототип,

створений у межах даної магістерської роботи, на повноцінний інструмент підтримки ухвалення рішень у сфері управління складом.

РОЗДІЛ 4. ПЕРСПЕКТИВИ РОЗВИТКУ

4.1 Можливості розширення моделі

Побудована симуляційна модель може бути розширена для підтримки ширшого спектра складських систем і стратегій. Зокрема, доцільно реалізувати такі можливості:

– **Інші типи складів:** підтримка автоматизованих систем зберігання і пошуку (AVS/RS) з багаторівневими шатлами і ліфтами, а також конвеєрних ліній для транспортування товарів. Це вимагатиме додавання нових типів ресурсів (ліфти, конвеєри) і логіки їх керування.

– **Альтернативна логіка пікінгу:** впровадження різних стратегій відбору товарів, наприклад, зональний відбір (коли кожен пікер закріплений за певною зоною) чи груповий відбір (batch picking). Це дозволить моделювати ще складніші сценарії організації роботи персоналу.

– **Черги на ремонт і бригади техобслуговування:** якщо додати моделювання відмов техніки, можна деталізувати процес ремонту, ввівши чергу на ремонт AGV і декілька ремонтних бригад. Тоді система зможе відображати ситуації, коли одночасно ламаються кілька роботів і частина чекає ремонту.

– **Вплив на сервісні метрики:** можна розширити модель, включивши показники рівня сервісу, наприклад SLA (частка замовлень, виконаних вчасно). Для цього ввести дедлайн на виконання кожного замовлення і обчислювати штрафи за запізнення. Це дозволить досліджувати компроміс між вартістю і якістю сервісу.

– **Запізнення та штрафи:** моделювання фінансових наслідків запізнень (penalty cost), що включається у вартісну модель. Це пов’язано з попереднім пунктом і дасть змогу оптимізувати не тільки час, а й витрати.

– **Теплові карти завантаження графа:** візуальне відображення, які маршрути і зони складу найчастіше використовуються, у вигляді “теплової карти”

інтенсивності руху AGV. Це допоможе ідентифікувати можливі “вузькі місця” у плануванні складу (наприклад, занадто вузькі проходи, де постійно їздять роботи).

– **Агреговані показники по зонах:** якщо склад поділений на секції, можна збирати статистику окремо по кожній зоні (скільки замовлень відпрацьовано, середній час по зоні тощо). Це корисно для оцінки, чи є дисбаланс у завантаженні різних частин складу.

4.2 Підтримка різних типів техніки

Модель можна доповнити іншими типами автоматизованої техніки, такими як стаціонарні роботи-пікери, роботизовані сортувальники, дрони для інвентаризації тощо. Наприклад, впровадивши роботизовані сортувальні станції на виході, можна моделювати, як вони впливають на throughput системи (ймовірно, знімаючи навантаження з пікерів, але додаючи свої черги).

Для кожного нового типу ресурсу знадобиться: - Додати клас ресурсу (з поведінкою в SimPy, каналами обслуговування). - Включити його у модель процесу (наприклад, після відбору може бути стадія сортування). - Налаштувати політики управління цим ресурсом (як призначаються завдання сортувальнику, чи є пріоритети).

Завдяки модульній архітектурі додавання нового ресурсу зводиться до імплементації його процесу і інтеграції у диспетчер або окремий диспетчер для нього.

4.3 Надійність та відмовостійкість

Хоча модель має спрощене уявлення про відмови (параметри MTBF/MTTR для AGV), її можна зробити детальнішою. Зокрема, реалізувати: - Різні типи відмов (механічна поломка, розряд батареї, помилка навігації тощо) з різними наслідками. - Модель деградації продуктивності (AGV може працювати повільніше при низькому

заряді). - Алгоритми predictive maintenance – коли модель намагається запланувати профілактичний ремонт, щоб уникнути раптових відмов у піковий час.

Дослідження показали, що відмови техніки можуть сильно впливати на продуктивність складу, особливо якщо немає резервних роботів[18]. Тому інтеграція модуля надійності розширить застосування симуляції для задач аналізу ризиків та оцінки необхідності резервування обладнання.

4.4 Інтеграція з реальними системам

Модель можна інтегрувати з реальними WMS/ERP системами для отримання даних в режимі реального часу та перевірки what-if сценаріїв на “цифровому двійнику” складу. Наприклад, підключившись до API WMS, симуляція могла б отримувати актуальний стан складу і прогнозувати наслідки прийняття певних рішень (наприклад, зміни пріоритетів замовлень). Цей напрям відомий як **цифровий двійник складу**[19][20].

Інтеграція вимагатиме: - Розробити API для обміну даними (наприклад, REST або gRPC сервіс, через який WMS надсилає дані про чергу, а симуляція повертає прогнози). - Забезпечити відображення поточного стану – ініціалізувати модель фактичними даними (поточні черги, розташування роботів). - Мати можливість швидко прокручувати симуляцію (наприклад, пришвидшений режим), щоб видавати прогнози на кілька годин вперед за частки секунди реального часу.

Це значно розширить практичну цінність симулятора: він зможе працювати як система підтримки прийняття рішень у режимі online.

4.5 Масштабування на більші системи

Для моделювання великих складів (десятки роботів, тисячі замовлень на день) може знадобитися оптимізація продуктивності симуляції. Окрім розглянутих у розд. 3.3 методів (Cython, багатопроцесність), можна використати спеціалізовані рішення:

– **SimPyRT** або інші розширення SimPy для реального часу.

– **Використання GPU** для паралельного виконання незалежних процесів (менш актуально, оскільки процеси в нашій моделі не однотипні масово).

– **Агреговані моделі**: замість моделювання кожного замовлення окремо – об’єднувати їх у пакети, якщо це не знижує точність результатів, що зменшить кількість подій.

Також при збільшенні масштабів варто приділити увагу ефективності коду Python: профілювати програму для виявлення “вузьких місць” і оптимізувати алгоритми і структури даних (наприклад, використовувати numpy для роботи з масивами).

4.6 Розширення інтерфейсу користувача

На останньому етапі розвитку проєкту доцільно удосконалити користувацький інтерфейс (UI) для підвищення зручності роботи з симулятором та наочності результатів. Перспективи розвитку UI охоплюють:

– **Сценарне моделювання:** забезпечити можливість створювати, зберігати і завантажувати різні сценарії моделювання через інтерфейс. Користувач повинен мати змогу налаштовувати параметри складу (кількість техніки, структуру, політики) та зберігати ці налаштування як окремі сценарії. Це спростить проведення серії експериментів та порівняння їх результатів.

– **Мультисценарний аналіз:** одночасне відкриття та порівняння кількох сценаріїв. Наприклад, UI може дозволяти запускати два сценарії паралельно та виводити їх ключові метрики поруч. Такий функціонал допоможе швидко ідентифікувати, який із сценаріїв є кращим за тими чи іншими показниками.

– **Візуалізація графа складу:** інтерактивне відображення топології складу – місць зберігання, зон, маршрутів руху AGV. Інтерфейс може мати режим візуалізації плану складу, де анімовано показано переміщення AGV та пікерів у реальному часі симуляції. Це зробить симулятор більш наочним і полегшить виявлення проблем (наприклад, скупчення AGV у певній зоні).

– **Аналітичні панелі:** інтеграція спеціалізованих візуальних компонентів для аналізу KPI. Наприклад, UI може містити панель з основними KPI (throughput, середній час циклу, поточний розмір черги тощо) в режимі реального часу під час симуляції. Також можна реалізувати інтерактивні графіки: гістограми розподілу часу виконання замовлень, діаграми завантаженості ресурсів, heatmap по графу руху. Це дасть можливість користувачу самостійно досліджувати “вузькі місця” через інтерактивний аналіз даних симуляції.

– **Експорт результатів:** забезпечити функції експорту результатів моделювання (у вигляді звітів, таблиць, графіків) для подальшого використання. Наприклад, менеджер може прогнати кілька сценаріїв, експортувати порівняльні показники у CSV чи PDF-звіт і представити їх для ухвалення рішень. UI має спростити цей процес, надаючи готові шаблони звітів.

– **Дружність до користувача та дизайн:** поліпшення структури інтерфейсу, додавання контекстної допомоги, підказок щодо налаштування параметрів моделі. Оскільки модель може мати багато параметрів, варто згрупувати їх по вкладках (наприклад, “Склад”, “Ресурси”, “Політики”, “Експерименти”). Інтуїтивно зрозумілий UI сприятиме ширшому застосуванню симулятора не тільки розробниками, а й інженерами складських операцій, аналітиками тощо.

Таким чином, розширення UI перетворить інструмент із прототипу на повноцінний програмний продукт, придатний для використання кінцевими користувачами. Зручний інтерфейс та широкі аналітичні можливості забезпечать ефективне застосування моделі у практиці управління складом.

ВИСНОВКИ

У магістерській роботі розроблено та досліджено імітаційну модель автоматизованого складу з використанням автономних мобільних роботів (AGV). Проведений аналіз літератури показав різноманіття сучасних технологій автоматизації складів та методів керування ними, що підтвердило актуальність вибраної теми. На основі цього аналізу сформовано концепцію системи моделювання складу, визначено її архітектуру та основні компоненти.

Реалізована дискретно-подієва симуляція складу охоплює весь цикл обробки замовлення – від надходження до виконання – та враховує ключові ресурси (AGV, пікери) і процеси. В моделі передбачено різні стратегії управління: політики диспетчеризації завдань для AGV, принципи відбору замовлень, режими роботи складу за профілем навантаження тощо. Це дозволило провести серію експериментів і оцінити вплив кожного фактору на продуктивність системи.

За результатами моделювання встановлено, що збільшення кількості AGV та оптимізація алгоритму призначення завдань здатні суттєво підвищити пропускну здатність складу та скоротити час циклу замовлення. Зокрема, використання оптимізаційного розподілу на основі угорського алгоритму (OR-Tools) перевершує жадібний підхід, зменшуючи простої техніки і час очікування замовлень. Профіль пікового навантаження значно впливає на динаміку черг: при інтенсивностях, що перевищують пропускну спроможність, спостерігається накопичення черги та зростання часу виконання, однак після завершення піку система поступово обробляє накопичені замовлення. Таким чином, модель адекватно відображає поведінку складу за різних сценаріїв навантаження.

Розроблена симуляційна модель може бути використана для подальших досліджень і вдосконалення роботи складу. Зокрема, її можна розширити для моделювання інших типів автоматизації (конвеєри, шатл-системи), дослідити вплив відмов техніки та стратегій технічного обслуговування, інтегрувати з реальними

даними WMS для створення “цифрового двійника” складу. Отримані в роботі результати та рекомендації можуть знайти практичне застосування при плануванні та експлуатації складів з високим рівнем автоматизації, допомагаючи приймати обґрунтовані рішення щодо інвестування в додаткову техніку або зміни організації процесів.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SimPy Discrete Event Simulation Framework. URL: https://simpy.readthedocs.io/
2. Google OR-Tools. Operations Research Tools. URL: https://developers.google.com/optimization
3. NetworkX: Network Analysis in Python. URL: https://networkx.org/
4. Pandas Documentation. URL: https://pandas.pydata.org/docs/
5. Matplotlib Documentation. URL: https://matplotlib.org/stable/contents.html
6. Streamlit Documentation. URL: https://docs.streamlit.io/
7. Python 3 Documentation. Python Software Foundation. URL: https://docs.python.org/3/
8. SimPy Tutorial: Discrete-Event Simulation in Python. URL: https://simpy.readthedocs.io/en/latest/simpy_intro/
9. Google Developers: Python OR-Tools Examples. URL: https://developers.google.com/optimization/examples
10. Warehouse Management System (WMS) – Overview. URL: https://www.sap.com/products/scm/warehouse-management.html
11. Gwynne Richards. Warehouse Management: A Complete Guide to Improving Efficiency and Minimizing Costs in the Modern Warehouse. – Kogan Page, 2017.
12. Bartholdi J. J., Hackman S. T. Warehouse & Distribution Science. – Release 0.98, 2019. – URL: https://www.warehouse-science.com/
13. Hopp W. J., Spearman M. L. Factory Physics. – 3rd ed. – McGraw-Hill, 2011.
14. Law A. M. Simulation Modeling and Analysis. – 5th ed. – McGraw-Hill, 2015.
15. Banks J. Discrete-Event System Simulation. – Prentice Hall, 5th ed., 2010.
16. Kershaw G. Automated Guided Vehicles: A Systems Approach. – CRC Press, 2018.

17. Mourtzis D. Simulation in the Design and Operation of Manufacturing Systems. – Springer, 2021.

18. Vis I. F. A. Survey of Research in the Design and Control of Automated Guided Vehicle Systems. – European Journal of Operational Research, 2006.

19. Gu J., Goetschalckx M., McGinnis L. F. Research on Warehouse Operation: A Comprehensive Review. – European Journal of Operational Research, 2007.

20. Roodbergen K. J., Vis I. F. A. A Survey of Literature on Automated Storage and Retrieval Systems. – European Journal of Operational Research, 2009.

21. Rushton A., Croucher P., Baker P. The Handbook of Logistics & Distribution Management. – Kogan Page, 2017.

22. Кіндрат П. В., та ін. Інформаційні системи управління в логістиці: навч. посіб. – Рівне: Острозька академія, 2020.

23. Григор’єв М. Н. Логістика складських процесів. – Київ: КНЕУ, 2018.

24. Босовець Г. Г. Імітаційне моделювання систем масового обслуговування. – Львів: Видавництво ЛНУ, 2016.

25. Імітаційне моделювання систем та процесів кібербезпеки в середовищі MATLAB: Практикум [Електронний ресурс] / [А.Д. Кожухівський, Г.І. Гайдур, О.А. Кожухівська, В.В. Марченко, С.О. Алексенко]; М-во освіти і науки України, Державний університет телекомунікацій. – К: ДУТ, 2020. – 78 с.

26. Jain R. The Art of Computer Systems Performance Analysis. – Wiley, 1991.

27. Gross D., Shortle J., Thompson J., Harris C. Fundamentals of Queueing Theory. – 4th ed. – Wiley, 2008.

28. Shanthikumar J. G., Buzacott J. A. Stochastic Models of Manufacturing Systems. – Prentice Hall, 1993.

29. Silver E. A., Pyke D. F., Peterson R. Inventory Management and Production Planning and Scheduling. – Wiley, 1998.

30. Hoen K. M. R., Valentini F. AGV Systems in Modern Warehouses: A Review of Planning and Control Issues. – International Journal of Production Research, 2019.

31. Plotly Python Graphing Library. URL: https://plotly.com/python/

32. Jupyter Project Documentation. URL: https://jupyter.org/

33. Docker Documentation. URL: https://docs.docker.com/

34. GitHub: Best Practices for Version Control in Data Science. URL: https://docs.github.com/

35. ISO 9001:2015 Quality Management Systems – Requirements. – ISO, 2015.

36. IEC 62264-1: Enterprise-control system integration – Part 1: Models and Terminology. – IEC, 2013.

37. OMG Unified Modeling Language (UML) Specification. URL: https://www.omg.org/spec/UML/

38. Марченко О. В. Моделювання та оптимізація логістичних систем: навч. посіб. – Київ: НАУ, 2019.

39. ДСТУ ISO 5807:2005 Документація інформаційних систем та технологій. Умовні познаки та умовні зображення. – Київ: Держспоживстандарт України, 2005.

40. ДСТУ 8302:2015 Бібліографічні посилання. Загальні положення та правила складання. – Київ: ДП «УкрНДНЦ», 2016.