Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Комп'ютерно-інформаційних систем і програмної інженерії
(повна назва факультету)
Програмної інженерії
(повна назва кафедри)

КВАЛІФІКАЦІЙНА РОБОТА

на здобуття освітнього ступеня

бакалавр
(назва освітнього ступеня)

на тему: «Розробка програмного забезпеченя симуляції міського середовища на основі мультиагентного підходу»

Виконав(ла): студент(ка) IV курсу, групи СП-42
спеціальності 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

(підпис) Штокало А. Р. (прізвище та ініціали)

Керівник (підпис) Тимків П. О. (прізвище та ініціали)

Нормоконтроль (підпис) Стоянов Ю. М. (прізвище та ініціали)

Завідувач кафедри (підпис) Петрик М. Р (прізвище та ініціали)

Рецензент (підпис) Млинко Б. Б. (прізвище та ініціали)

Тернопіль
2026

Міністерство освіти і науки України
Тернопільський національний технічний університет імені Івана Пулюя

Факультет комп'ютерно-інформаційних технологій і програмної інженерії
(повна назва факультету)

Кафедра програмної інженерії
(повна назва кафедри)

ЗАТВЕРДЖУЮ
Завідувач кафедри
проф. Петрик М. Р.
(підпис) (підпис)
« 6 » квітня 2026 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

на здобуття освітнього ступеня бакалавр
(назва освітнього ступеня)

за спеціальністю 121 «Інженерія програмного забезпечення»
(шифр і назва спеціальності)

студенту Штокало Андрію Романовичу
(прізвище, ім'я, по батькові)

1. Тема роботи «Розробка програмного забезпечення симуляції міського середовища на основі мультиагентного підходу»

Керівник роботи доцент кафедри БТ Тимків П. О.
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Затверджені наказом ректора від « 06 » квітня 2026 року № _____
2. Термін подання студентом завершеної роботи 22.06.2026
3. Вихідні дані до роботи наукові літературні джерела

4. Зміст роботи (перелік питань, які потрібно розробити)
Вступ. 1 Аналіз предметної області та постановка задачі.
2. Проєктування тпрограмної системи 3. Реалізація програмної системи.
4. Тестування, впровадження та аналіз результатів.
5. Безпека життєдіяльності, основи охорони праці.
Висновки.
5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень, слайдів)
1. Тема роботи. 2. Актуальність, мета, задачі дослідження
3. Існуючі технології реалізації подібних систем.
4. Функціональні та нефункціональні вимоги .5. Загальна архітектура системи.
6. Варіанти використання. 7. Компоненти програми для налаштування параметрів системи.
8. Програмні засоби та технології. 9. Інтерфейси реалізації застосунку..
10. Тестування. 11. Висновки по роботі. 12. Слайди презентації.

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Безпека життєдіяльності, основи охорони праці			

7. Дата видачі завдання _____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів роботи	Термін виконання етапів роботи	Примітка
1.	*Розробка технічного завдання*	*6.04 – 12.04*	Виконано
2.	*Робота над першим розділом «Аналіз предметної області та постановка задачі»*	*13.04 – 26.04*	Виконано
3.	*Робота над другим розділом «Проєктування програмної системи»*	*27.04 – 03.05*	Виконано
4.	*Робота над третім розділом «Реалізація програмної системи»*	*04.05 – 10.05*	Виконано
5.	*Робота над четвертим розділом «Тестування, впровадження та аналіз результатів»*	*11.05 – 17.05*	Виконано
6.	*Робота над п'ятим розділом «Безпека життєдіяльності, основи охорони праці»*	*18.05 – 24.05*	Виконано
7.	*Оформлення пояснювальної записки і графічного матеріалу*	*25.05 – 7.06*	Виконано
8.	*Перевірка на академічний плагіат, перевірка керівником та консультантами*	*8.06 – 14.06*	
9.	*Попередній захист кваліфікаційної роботи бакалавра*	*15.06 – 21.06*	
10.	*Захист кваліфікаційної роботи бакалавра*		

Студент _____ Штокало А. Р.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Тимків П. О.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Розробка інтелектуальної системи симуляції міського середовища на основі мультиагентного підходу // Кваліфікаційна робота освітнього рівня «Бакалавр» // Штокало Андрій Романович // Тернопільський національний технічний університет імені Івана Пулюя, факультет комп'ютерно-інформаційних систем, кафедра програмної інженерії, група СП-42 // Тернопіль, 2026 // С. 65, рис. – 18, додат. – 6, бібліогр. – 29.

Ключові слова: мультиагентна система, міська симуляція, агент, потреби агента, тайлова карта, журнал рішень.

Кваліфікаційна робота присвячена розробленню програмної системи мультиагентної симуляції міського середовища.

У першому розділі проаналізовано предметну область агентного моделювання міста, розглянуто наявні підходи та сформовано вимоги до програмної системи.

У другому розділі спроєктовано функціональну, класову, алгоритмічну та логічну моделі системи, а також визначено структуру бази даних.

У третьому розділі описано реалізацію системи Urban Multi-Agent Simulation, що включає генерацію населення, математичну модель вибору дій, A*-маршрутизацію, SQLite-сховище та клієнтську вебвізуалізацію результатів.

У четвертому розділі проведено тестування розробленого рішення, підготовлено аналітичні візуалізації результатів симуляції та наведено системні вимоги для її впровадження.

Об’єкт дослідження: процеси імітаційного моделювання поведінки агентів у міському середовищі.

Предмет дослідження: методи та програмні засоби побудови мультиагентної симуляції з тайловою картою та потребнісною моделлю агентів.

ABSTRACT

Development of an intelligent urban environment simulation system based on a multi-agent approach // Qualification work of the educational level «Bachelor» // Shtokalo Andrii Romanovych // Ternopil Ivan Puluj National Technical University, Faculty of Computer Information Systems, Software Engineering Department, group SP-42 // Ternopil, 2026 // P. 65, fig. – 18, annexes. – 6, references – 29.

Keywords: multi-agent system, urban simulation, agent, agent needs, tile map, decision log.

The qualification work is devoted to the development of a software system for multi-agent urban environment simulation.

The first chapter analyzes the subject area of urban agent-based modeling, considers existing approaches, and forms the requirements for the software system.

The second chapter designs the functional, class, algorithmic, and logical models of the system, and defines the database structure.

The third chapter describes the implementation of the Urban Multi-Agent Simulation system, including population generation, mathematical action selection model, A* routing algorithm, SQLite storage, and client-side web visualization.

The fourth chapter tests the developed solution, prepares analytical visualizations of the simulation results, and presents the system requirements for its implementation.

Object of research: processes of simulation modeling of agent behavior in an urban environment.

Subject of the study: methods and software tools for building a multi-agent simulation with a tile map and a needs-driven agent model.

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

Агент – програмна модель людини у міському середовищі. Агент має ідентифікатор, ім’я, вектор рис, вектор поточних потреб, позицію, домашню позицію та розклад.

Потреби агента – нормалізовані числові характеристики внутрішнього стану агента: енергія, ситість, щастя, соціальність, здоров’я, гроші, стрес і продуктивність.

A – алгоритм пошуку шляху на графі або сітці з урахуванням евристичної оцінки відстані до цілі.

Action Space – простір можливих дій агента. Кожна дія має категорію, цільову зону, тривалість, ефекти на потреби, вектор рис, часові обмеження та, за потреби, обов’язковий сервіс.

API – програмний інтерфейс взаємодії між клієнтською частиною вебзастосунку та серверною частиною системи.

Canvas – HTML-елемент для програмного малювання карти, агентів, зон, входів, теплової карти та траєкторій у браузері.

Decision Log – журнал прийняття рішень агентом. Він містить tick, agent_id, обрану дію, категорію, причину вибору, цільову координату, числову оцінку, пояснення та альтернативи.

JSON – текстовий формат подання структурованих даних, який використовується для опису карти, дій, агентів, сценаріїв і відповідей HTTP API.

Offcanvas – бокова висувна панель інтерфейсу, у якій після вибору агента або зони показуються деталі, журнал, рішення, список мешканців зони або доступні дії.

Persona – статичний профіль агента, що визначає його схильність до певних типів дій. У системі використано риси economy, social, activity і diligence.

SQLite – вбудована реляційна система керування базами даних, що використовується для збереження дій, агентів, симуляцій, timeline, decision logs і метрик.

Tick – дискретний крок модельного часу. У поточній реалізації один tick відповідає одній віртуальній хвилині.

Tiled – редактор тайлових карт, формат якого використано для опису міського середовища у вигляді ортогональної сітки з семантичними властивостями тайлів.

Timeline – попередньо обчислений запис симуляції, який містить послідовність кадрів. Кожен кадр зберігає стан усіх агентів у конкретний tick.

ЗМІСТ

ВСТУП

Сучасне міське середовище є складною системою, у якій поведінка людей залежить від потреб, просторових обмежень, розкладу, доступності зон, стану інфраструктури та зовнішніх сценарних умов. Для аналізу таких процесів недостатньо статичних карт або простих анімацій руху точок. Потрібна модель, у якій кожний учасник має власний стан, приймає рішення, переміщується за правилами середовища та залишає простежуваний запис своїх дій.

Актуальність роботи зумовлена потребою у прозорій і відтворюваній системі для демонстрації поведінки агентів у місті. Багато професійних симуляторів є складними для швидкого розгортання або орієнтовані на специфічні задачі транспортного моделювання. У межах цієї роботи розробляється компактна система, у якій основну увагу приділено пішохідній поведінці агентів, логіці входу до міських зон, поясненню рішень, збереженню timeline та інтерактивному аналізу результатів.

Метою кваліфікаційної роботи є розроблення програмної системи мультиагентної симуляції міського середовища, яка забезпечує генерацію населення, математично обґрунтований вибір дій агентами, маршрутизацію по пішохідній мережі, збереження повного запису симуляції та браузерне відтворення результатів.

Для досягнення мети потрібно виконати такі задачі: проаналізувати предметну область агентного моделювання міста; сформувати вимоги до програмної системи; спроєктувати карту, агентів, дії, сценарії, базу даних і вебінтерфейс; реалізувати модель агента з потребами, рисами та розкладом; реалізувати вибір дій на основі utility-оцінки; реалізувати маршрутизацію по тайловій карті; реалізувати генерацію агентів із керованою випадковістю; реалізувати збереження timeline, decision logs, agent journals і метрик; реалізувати вебплеєр із масштабуванням, перетягуванням карти, вибором агентів і зон; підготувати експорт даних для аналітичних діаграм; провести тестування системи.

Об’єктом дослідження є процеси імітаційного моделювання поведінки агентів у міському середовищі.

Предметом дослідження є методи та програмні засоби побудови мультиагентної симуляції з тайловою картою, потребнісною моделлю агентів, алгоритмічною маршрутизацією, збереженням запису симуляції та вебвізуалізацією.

Науково-прикладна новизна полягає в адаптації підходів міського агентного моделювання до компактної системи, у якій рішення агентів є пояснюваними, числовими та придатними для автоматизованої перевірки. На відміну від випадкового руху або непрозорої поведінкової генерації, у розробленій системі кожна дія агента пов’язана з його потребами, розкладом, persona, доступністю дії та просторовою ціллю.

Практичне значення результатів полягає у створенні MVP вебсистеми, яку можна використовувати як навчальний, демонстраційний і дослідницький інструмент для аналізу агентної поведінки в місті. Система дозволяє змінювати кількість агентів, тривалість, seed і сценарій середовища, після чого отримувати відтворюваний запис симуляції, маршрути, журнали рішень і агреговані метрики.

Апробація результатів роботи здійснювалася шляхом підготовки тапублікації тез наукової конференції за тематикою кваліфікаційної роботи.Матеріали публікації наведено в додатку А.

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА ПОСТАНОВКА ЗАДАЧІ

У цьому розділі описано предметну область мультиагентного моделювання міського середовища, проаналізовано споріднені підходи, сформовано вимоги до програмної системи та визначено постановку задачі. Структуру основної частини роботи узгоджено з методичними рекомендаціями до виконання кваліфікаційної роботи бакалавра [1].

1.1 Аналіз предметної області

Міське середовище буде розглядатися як сукупність функціональних зон, пішохідної інфраструктури, просторових обмежень, сервісів і агентів, які взаємодіють із цим середовищем. У реальному місті люди не переміщуються довільно крізь будівлі, дороги або перешкоди. Вони користуються тротуарами, переходами, входами до будівель, робочими місцями, навчальними закладами, зонами відпочинку, комерційними просторами та медичними пунктами. Тому симуляція міського середовища повинна відображати не тільки координати агентів, а й логіку доступності.

У межах роботи буде використано тайлове подання міста. Місто буде розбите на ортогональну сітку, де кожний тайл має координати, тип зони, ознаку прохідності, вартість проходу, сервіси та властивості входу до зони. Такий підхід дозволяє просторово відокремити дороги, тротуари, переходи та функціональні зони. Дороги у розробленій системі будуть виконувати роль міського бар’єра та декоративної інфраструктури, але агенти як пішоходи не зможуть ними пересуватися. Переміщення між зонами буде можливим тільки по тротуарах і пішохідних переходах.

Поведінка агента буде залежати від внутрішнього стану. Стан буде описано вектором потреб: енергія, ситість, щастя, соціальність, здоров’я, гроші, стрес і продуктивність. Усі значення будуть нормалізовані в діапазоні від нуля до одиниці. Низькі значення енергії, ситості, щастя, соціальності, здоров’я, грошей і

продуктивності означатимуть потребу у відновленні відповідного параметра. Для стресу інтерпретація буде протилежною: більше значення означатиме гірший стан.

1.2 Аналіз наявних підходів і споріднених систем

У споріднених роботах міська поведінка часто моделюється через поєднання просторової структури міста, потреб агентів, звичок, обов'язкових дій і статистичного аналізу результатів. У роботі MobileCity використано міську симуляцію з агентами, які мають потреби, звички й обов'язки, а результати подано через діаграми зайнятості локацій, зміни потреб і розподілу активностей [2].

Відкритий репозиторій MobileCity буде використано як концептуальний орієнтир для розуміння структури міської карти, запису симуляцій і подання результатів. У власній роботі не копіюється архітектура MobileCity, але зберігається корисна ідея: міська симуляція повинна мати не лише візуальне відтворення, а й дані для аналітики поведінки [3].

Мультиагентні системи як клас програмних систем дозволяють моделювати багато автономних сутностей, кожна з яких має власний стан і правила прийняття рішень. Для цієї роботи суттєвим є не максимальна складність агента, а здатність описати багато індивідуальних об'єктів через єдину формальну модель і потім отримати агреговані патерни з їхніх локальних рішень [4].

Для задач географічного та міського аналізу агентно-орієнтоване моделювання часто поєднується з просторовими структурами, картами, зонами та обмеженнями руху. Саме тому у роботі буде використано тайлову карту, entrances, travel cost і відокремлення пішохідних маршрутів від транспортних доріг [5].

Генеративні агенти можуть формувати правдоподібну поведінку і природні пояснення, але такі підходи часто потребують складної пам'яті, мовних моделей і значних обчислювальних ресурсів, тому доцільніше використати прозору числову модель, яку легше тестувати, пояснювати, відтворювати та розгортати локально [6].

Статистичні моделі потоків зручні для швидкого оцінювання великих потоків, але вони не показують індивідуальних маршрутів і журналів рішень.

Класичні мультиагентні моделі краще підходять для цієї роботи, тому що дозволяють формалізувати автономних агентів, просторові обмеження та правила поведінки. Генеративні агенти є перспективними для природної поведінки, але у даних межах вони надмірні за складністю. MobileCity-подібний підхід є найближчим концептуально, бо поєднує карту, потреби, обов'язки й аналітичні діаграми. Розроблена система займає проміжну позицію: вона не використовує LLM, але реалізує пояснювану числову поведінку, маршрутизацію, timeline, SQLite-запис, вебвідтворення та експорт аналітики.

1.3 Формування вимог до програмної системи

За результатами аналізу було сформовано вимоги до системи. Основна вимога полягає в тому, що система повинна демонструвати не випадкове переміщення агентів, а логічну поведінку, пов'язану з потребами, розкладом, зонами та маршрутами. Для пошуку маршрутів у сітковому середовищі буде використано алгоритм A*, формальний принцип якого описано у класичній роботі про евристичне визначення шляху з мінімальною вартістю [7].

Функціональні вимоги до системи такі:

– система повинна читати тайлову карту з JSON-даних і отримувати з неї розмір, типи зон, сервіси, входи, travel cost і прохідність;

– система повинна створювати задану кількість агентів на основі архетипів, підтримувати seed і обмежувати кількість агентів у безпечному діапазоні;

– агенти повинні розподілятися між кількома житловими зонами, а не концентруватися в одному будинку без потреби;

– система повинна оновлювати потреби агентів на кожному tick і не дозволяти значенням виходити за нормалізований діапазон;

– агент повинен обирати дію за потребами, розкладом, часовими вікнами, відповідністю persona і вартістю дії;

– рішення агента повинно записуватися у decision log з причиною, оцінкою, ціллю і альтернативами;

– агент повинен рухатися між зонами тільки по тротуарах і пішохідних переходах;

– агент повинен входити до зони через позначений entrance tile, який торкається потрібної функціональної зони;

– усередині зони агент повинен переміщуватися вільно, без жорсткої прив’язки до сітки, щоб кілька агентів не накладалися один на одного без потреби;

– система повинна зберігати повний timeline, decision logs, metrics і agent journals у SQLite;

– користувач повинен мати змогу програвати симуляцію у браузері, керувати timeline, масштабувати карту, перетягувати її, вибирати агента або зону;

– користувач повинен мати змогу створювати нові симуляції через модальне вікно, налаштовувати кількість агентів, час, тривалість, seed і сценарій;

– система повинна підтримувати видалення симуляцій, експорт JSON/CSV та підготовку CSV-даних для дипломних діаграм.

Нефункціональні вимоги також суттєві для якості MVP:

– система повинна бути відтворюваною: однакові seed, карта, набір дій, стартовий час і кількість агентів повинні давати стабільну структуру населення та близьку поведінкову динаміку;

– система повинна бути підтримуваною: карта, дії, сценарії та архетипи агентів мають зберігатися окремо від коду;

– система повинна бути достатньо продуктивною для 100 агентів і 720-1440 tick;

– система повинна запускатися локально через Python або Docker Compose;

– система повинна мати тести для основних модулів.

1.4 Актори, варіанти використання та постановка задачі

Основним актором системи буде користувач. Він запускає застосунок, генерує симуляцію, задає параметри, програє timeline, вибирає агентів і зони, аналізує рішення, переглядає метрики, експортує дані та видаляє непотрібні записи. Сервер симуляції є системним актором, який обробляє запити, запускає рушій і зберігає дані. SQLite-сховище є зовнішнім щодо бізнес-логіки ресурсом, який забезпечує довготривале збереження.

Ключові варіанти використання системи такі: створити нову симуляцію; вибрати наявну симуляцію; програти або поставити на паузу timeline; перемотати timeline; наблизити або віддалити карту; перетягнути карту; вибрати агента; переглянути журнал агента; вибрати зону; переглянути інформацію про зону; переглянути метрики; експортувати JSON або CSV; видалити симуляцію; підготувати дані для дипломних діаграм.

Постановка задачі формулюється так. Необхідно розробити програмну систему, яка моделює множину агентів $A = \{a_1, a_2, \dots, a_n\}$ у міському середовищі M, поданому тайловою картою. Для кожного агента a_i у кожний момент модельного часу t потрібно визначити позицію $p_i(t)$, вектор потреб $N_i(t)$, стан $s_i(t)$, активну дію $u_i(t)$, маршрут до цілі та журнал прийнятих рішень. Система повинна зберегти послідовність кадрів $F = \{F_0, F_1, \dots, F_T\}$, де кожний кадр містить стан усіх агентів.

2 ПРОЄКТУВАННЯ ПРОГРАМНОЇ СИСТЕМИ

Проєктування програмної системи виконано як сукупність взаємопов’язаних моделей, що описують функції системи, структурні елементи, сценарії взаємодії, алгоритмічні процеси, стани об’єктів та логічну організацію даних.

У межах проєктування було побудовано функціональну діаграму користувацьких сценаріїв, діаграму класів модельованої програмної системи, діаграми послідовностей для основних сценаріїв, діаграми діяльності для складних процесів, діаграми станів для агента й симуляції, а також логічну модель SQLite-сховища.

2.1 Функціональна модель системи

Функціональна модель системи описує взаємодію користувача з програмним продуктом. Користувач створює симуляції, задає їх параметри, переглядає список збережених записів, запускає відтворення timeline, перемотує модельний час, змінює масштаб карти, вибирає агентів і зони, переглядає журнали рішень, аналізує метрики, експортує дані та видаляє непотрібні симуляції. Серверна частина системи виконує обробку запитів, генерацію симуляцій, збереження результатів і повернення готових записів через HTTP API. SQLite-сховище забезпечує довготривале збереження timeline, журналів і агрегованих метрик.

Модель варіантів використання відображає головні межі системи. Центральним сценарієм є створення симуляції, оскільки саме він об’єднує налаштування параметрів, завантаження карти, генерацію населення, запуск рушія та збереження результату. Сценарій перегляду симуляції охоплює відтворення timeline, вибір агента, вибір зони, перегляд offcanvas-панелей і роботу з глобальними метриками. Сценарій експорту забезпечує подальший аналіз результатів у вигляді JSON або CSV.

Діаграму варіантів використання наведено на рисунку 2.1.

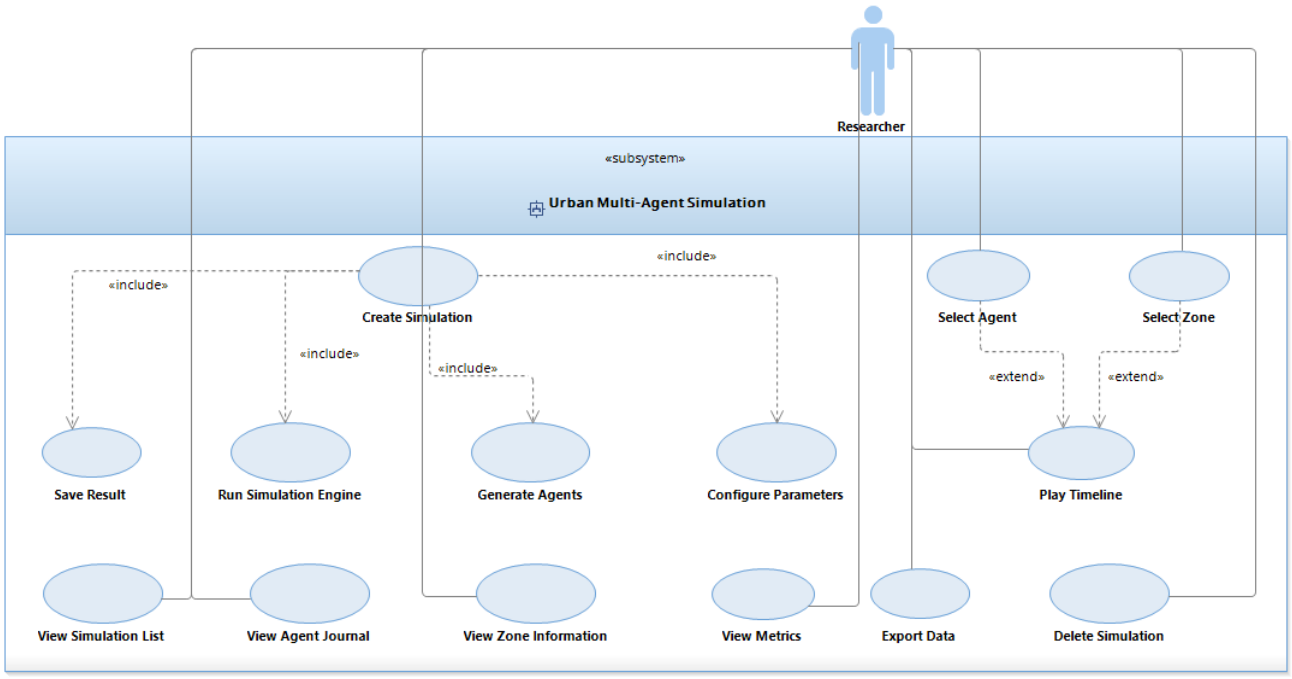


Рисунок 2.1 – Діаграма варіантів використання програмної системи

2.2 UML-діаграма класів модельованої програмної системи

Для опису структури модельованої програмної системи побудовано UML-діаграму класів зображену на рисунку 2.2:

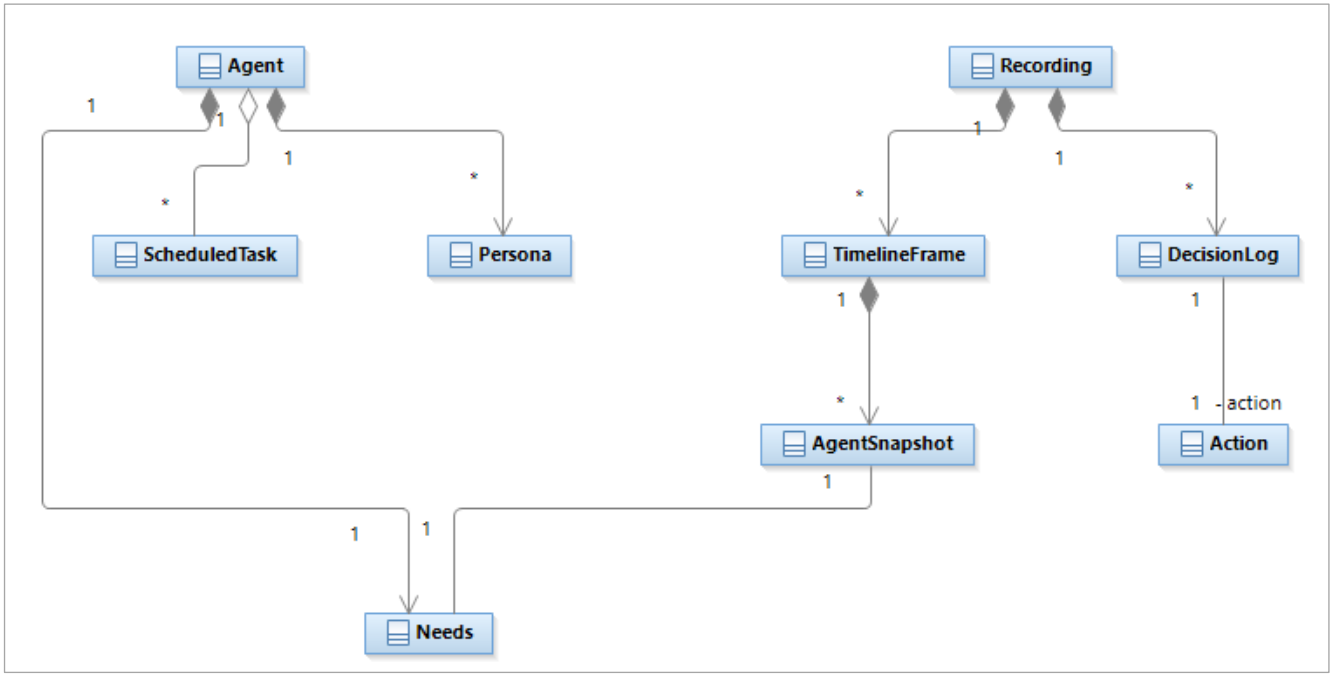


Рисунок 2.2 – UML-діаграма класової структури системи

Клас Needs описує поточний стан потреб агента. Він містить значення енергії, голоду, щастя, соціальності, здоров’я, грошей, стресу та продуктивності. Методи класу використовуються для поступової зміни потреб під час симуляції, застосування ефектів виконаної дії, а також для перетворення об’єкта у словник і відновлення його зі словника.

Клас Persona задає індивідуальні риси агента. Він зберігає набір параметрів поведінки у вигляді словника traits. Ці параметри використовуються під час вибору дії, щоб агенти з різними рисами не поводилися однаково. Методи класу дають змогу отримати вектор ознак для вибраного набору ключів, а також зберігати й відновлювати об’єкт у структурованому вигляді.

Клас ScheduledTask описує заплановану дію агента. Він містить ідентифікатор дії, початковий tick і кінцевий tick. Основний метод класу перевіряє, чи активне це завдання в певний момент часу. Це потрібно для того, щоб агент міг виконувати не лише випадково вибрані дії, а й дії зі свого розкладу.

Клас Action описує дію, доступну агенту в симуляції. Він містить ідентифікатор, назву, категорію, цільову зону, потрібний сервіс, тривалість, вплив на потреби, вектор рис, часові обмеження та додаткові метадані. Метод available_at перевіряє, чи може дія виконуватися в конкретний tick. Методи перетворення в словник і відновлення зі словника потрібні для обміну даними між частинами системи.

Клас Agent представляє окремого агента симуляції. Він містить ідентифікатор, ім’я, об’єкти Persona і Needs, поточну позицію, домашню позицію та список запланованих завдань. Метод scheduled_action_id визначає, чи має агент заплановану дію в заданий момент часу. Також клас підтримує перетворення свого стану у словник і відновлення з нього.

Клас AgentSnapshot фіксує стан агента в одному кадрі симуляції. Він містить ідентифікатор агента, tick, координати на карті, координати для відображення, зону, стан агента, активну дію та поточні потреби. Цей клас потрібен для запису timeline, щоб вебінтерфейс міг відтворювати симуляцію після її завершення.

Клас TimelineFrame описує один кадр симуляції. Він містить номер tick і список об’єктів AgentSnapshot. Кожен такий кадр зберігає стан усіх агентів у певний момент часу. Послідовність кадрів утворює повний запис руху й поведінки агентів.

Клас DecisionLog зберігає інформацію про вибір дії агентом. Він містить tick, ідентифікатор агента, вибрану дію, категорію, причину вибору, цільову позицію, оцінку, пояснення та список альтернатив. Ці дані використовуються для аналізу того, чому агент обрав саме цю дію.

Клас Recording описує повний результат виконання симуляції. Він містить ідентифікатор симуляції, тривалість tick у секундах, список кадрів TimelineFrame, журнали рішень DecisionLog, метрики та журнали агентів. Саме цей клас об’єднує всі дані, потрібні для збереження, перегляду та аналізу симуляції.

Між класами встановлено зв’язки, які показують, які об’єкти входять до складу інших об’єктів або використовуються ними. Agent містить один об’єкт Needs і один об’єкт Persona. Також агент може мати нуль або більше запланованих завдань ScheduledTask. AgentSnapshot пов’язаний з Needs, оскільки зберігає стан потреб агента в конкретний момент часу. TimelineFrame складається з одного або кількох об’єктів AgentSnapshot. Recording містить послідовність кадрів TimelineFrame і може містити багато записів DecisionLog. DecisionLog пов’язаний з Action, оскільки фіксує, яку дію було вибрано.

Повну версію UML-діаграми з атрибутами, методами та кратністю зв’язків наведено в додатку Б.

2.3 Модель сценаріїв взаємодії

Сценарії взаємодії описано через діаграми послідовності, оскільки вони фіксують порядок повідомлень між об’єктами. Для системи було виділено три ключові сценарії: створення симуляції, перегляд агента та експорт аналітичних даних. Ці сценарії охоплюють основні шари продукту: вебінтерфейс, HTTP API, рушій симуляції, сховище, карту, сервіс вибору дій і модуль експорту.

У сценарії створення симуляції користувач задає параметри у модальному вікні, клієнтська частина надсилає запит до API, сервер завантажує карту та набір дій, генерує агентів, запускає рушій, формує timeline і журнали рішень, після чого зберігає запис у SQLite. Результатом сценарію є готовий simulation_id, який одразу доступний для відтворення у браузері.

Діаграму послідовності створення симуляції наведено на рисунку 2.4.

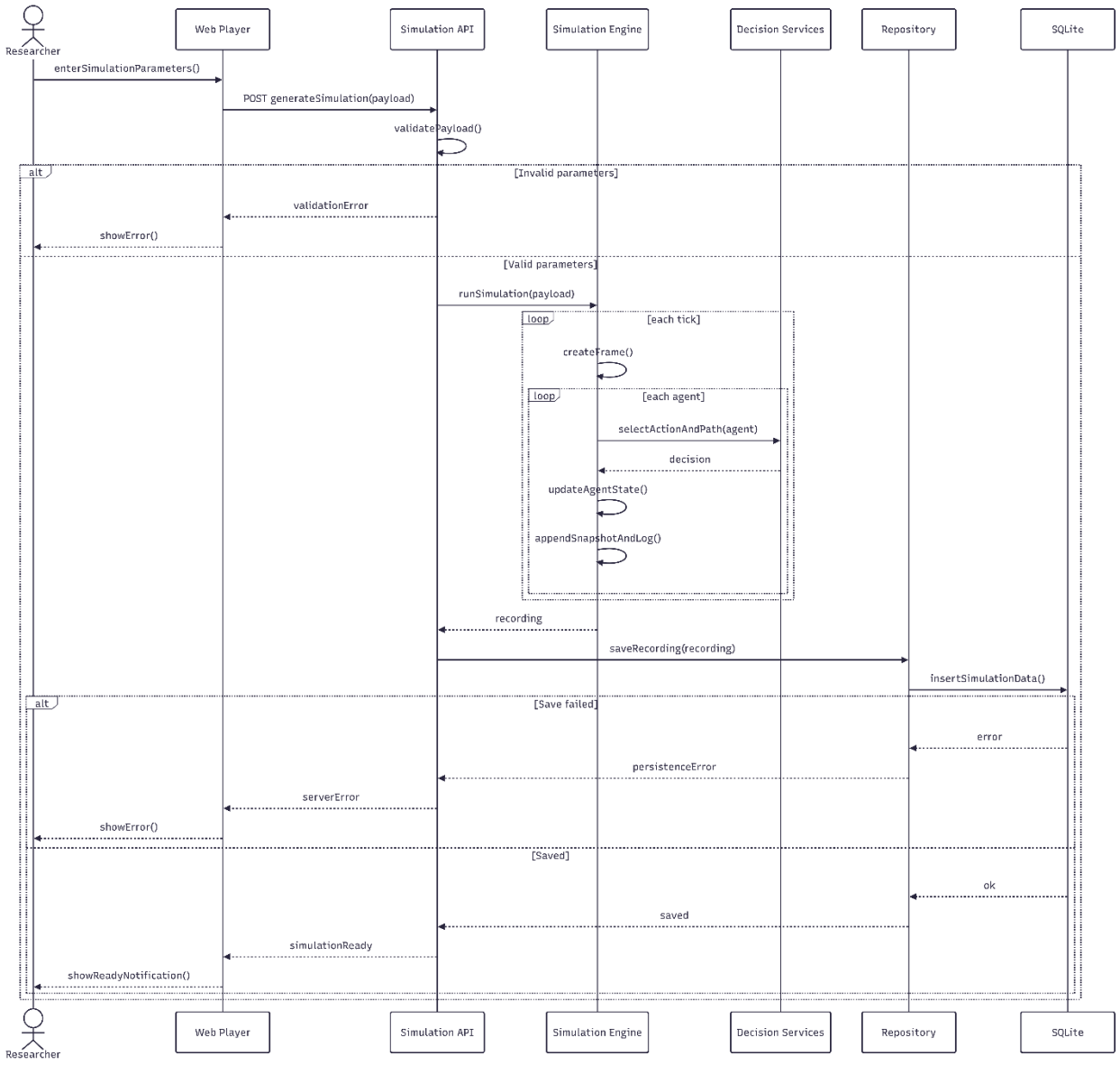


Рисунок 2.4 – Діаграма послідовності сценарію створення симуляції

У сценарії перегляду агента клієнтська частина працює переважно з уже завантаженим Recording. Після кліку по Canvas система визначає найближчого

агента, фіксує selectedAgentId, збирає фрагмент маршруту навколо поточного tick, вибирає відповідні записи DecisionLog і відкриває offcanvas із поточним станом, розкладом, рішенням, альтернативами та журналом. Це дає змогу аналізувати поведінку не тільки агреговано, а й на рівні конкретного агента.

Діаграму послідовності перегляду агента наведено на рисунку 2.5.

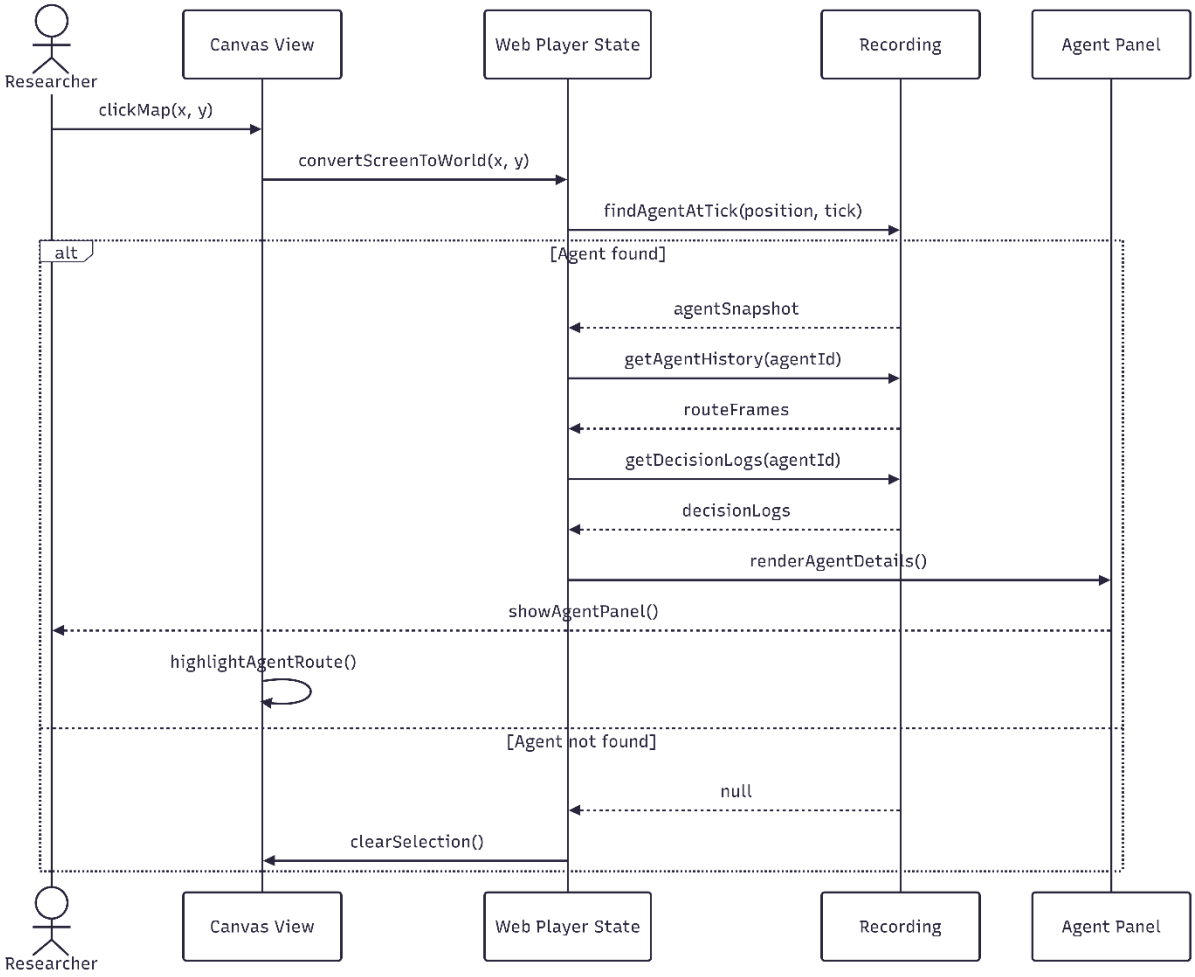


Рисунок 2.5 – Діаграма послідовності сценарію перегляду агента та журналу

Експорт аналітичних даних виділено окремим сценарієм, оскільки він перетворює повний запис симуляції на набори даних для діаграм. ChartExporter читає timeline, decision_logs і metrics зі сховища, агрегує зайнятість зон, динаміку потреб, розподіл активностей, переходи між зонами та підсумки маршрутів, після чого формує CSV-файли.

Діаграму послідовності експорту метрик наведено на рисунку 2.6.

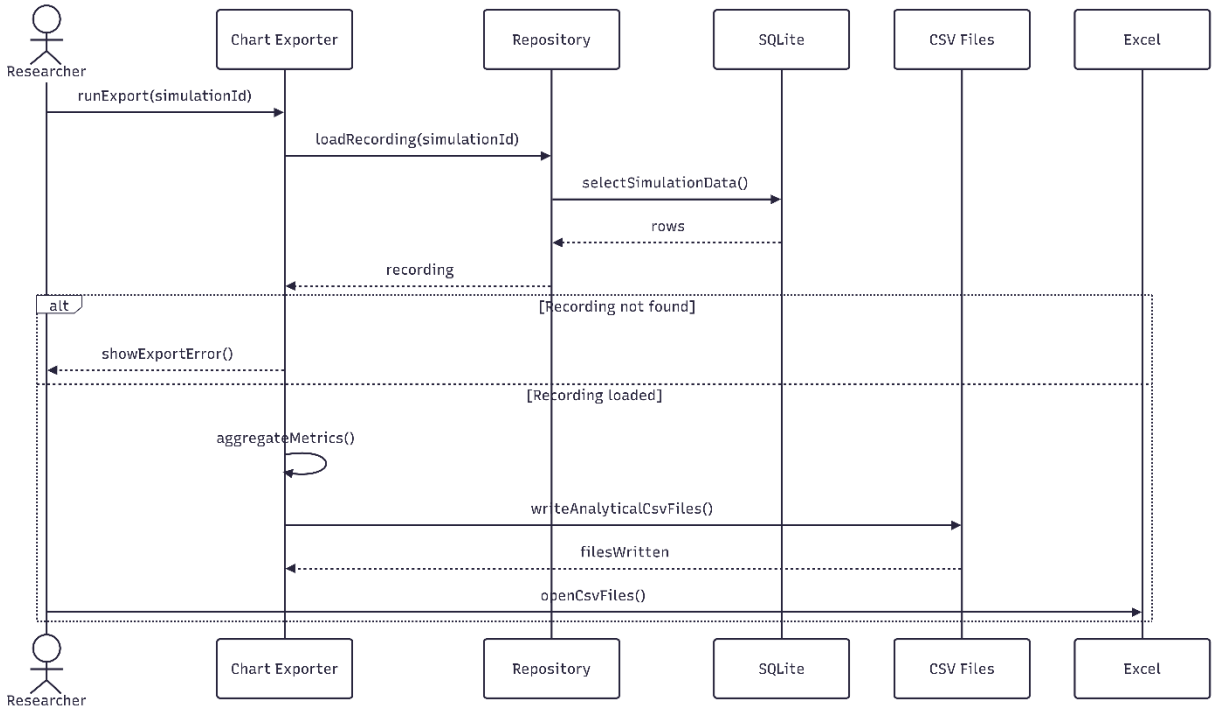


Рисунок 2.6 – Діаграма послідовності сценарію експорту аналітичних даних

2.4 Модель алгоритмічних процесів

Алгоритмічні процеси системи описано діаграмами діяльності. Вони показують не структуру класів, а перебіг дій, умови переходів і розподіл відповідальності між користувачем, вебінтерфейсом, API, рушієм та сховищем.

Процес генерації симуляції починається з введення параметрів користувачем і завершується збереженням готового Recording. У середині процесу відбувається перевірка параметрів, завантаження даних, генерація агентів, покрокове оновлення станів, вибір дій, побудова маршрутів, створення кадрів timeline, формування журналів і метрик (див. додаток В).

Процес перегляду timeline об’єднує завантаження запису, відображення карти, керування playback, інтерполяцію агентів, обробку кліків і оновлення інформаційних панелей. У цьому процесі важливо, що загальні метрики залишаються в основній боковій панелі, а деталі агента або зони відкриваються в offcanvas (див. додаток Д).

Процес підготовки аналітичних візуалізацій описує шлях від контрольної симуляції до готових рисунків. Він охоплює генерацію запису, експорт CSV,

перевірку колонок, побудову діаграм, оформлення легенд, підпис осей і вставлення рисунків у текст роботи. У системі цей процес винесено в окремий інструмент експорту, щоб аналітичні графіки можна було відтворити для різних seed і параметрів симуляції (див. додаток Е).

2.5 Модель станів агента і симуляції

Діаграми станів описують об’єкти, поведінка яких змінюється в часі. У розробленій системі такими об’єктами будуть runtime-стан агента та запис симуляції. Агент переходить між станами залежно від потреб, обраної дії, маршруту, доступності входу й завершення активності. Симуляція переходить між станами залежно від створення, збереження, відкриття, програвання, паузи, експорту та видалення.

Станова модель агента відображає повний цикл від ініціалізації до завершення симуляції. Агент може перебувати в очікуванні, виборі дії, русі до входу, вході в зону, внутрішньому перебуванні, виконанні дії, оновленні потреб або стані очікування через відсутній маршрут. Така модель пояснює, чому агент не рухається крізь стіни й чому внутрішній рух у зоні відокремлений від зовнішнього маршруту.

Діаграму станів агента наведено на рисунку 2.10.

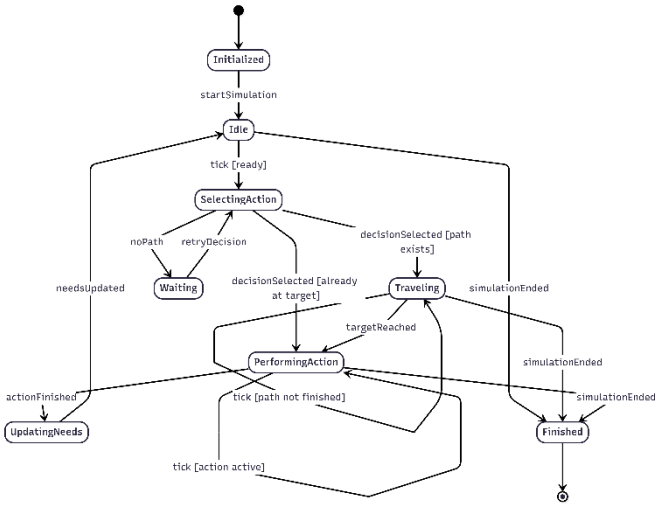


Рисунок 2.10 – Діаграма станів агента

Станова модель симуляції фіксує життєвий цикл запису. До генерації симуляція має стан Draft, під час обчислення – Generating, після успішного збереження – Ready. У браузері запис може перебувати у Playing або Paused. Під час експорту формується стан Exporting, після видалення – Deleted, а в разі помилки завантаження, генерації або запису – Error.

Діаграму станів симуляції наведено на рисунку 2.11.

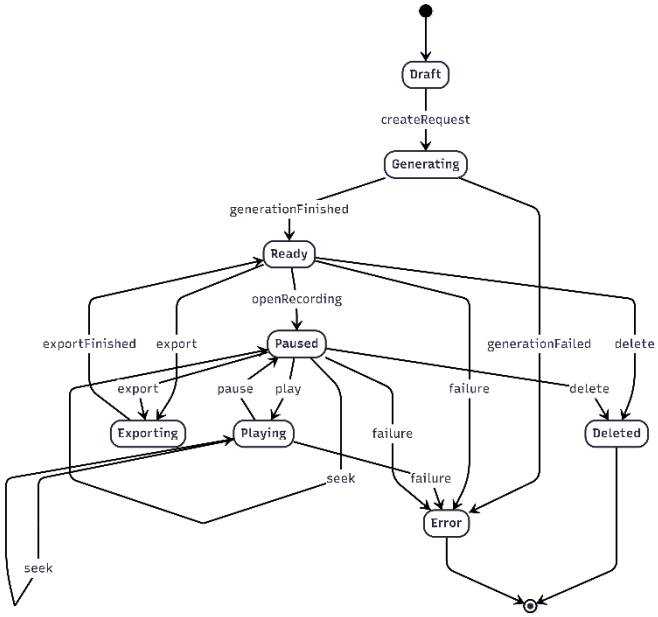


Рисунок 2.11 – Діаграма станів симуляції

2.6 Логічна модель даних

Логічна модель даних відображає, які сутності зберігаються у SQLite та як вони пов’язані між собою. Центральною сутністю є simulations, що ідентифікує окремий запис симуляції та зберігає параметри запуску. Сутність agents містить інформацію про агентів конкретної симуляції, їхні профілі, стартові потреби, домашні позиції та розклади. Timeline зберігає покадрові стани агентів, decision_logs фіксує обрані дії та альтернативи, simulation_metrics містить агреговані результати.

У моделі використано зв’язки один-до-багатьох від simulations до agents, timeline, decision_logs і simulation_metrics. Для timeline доцільним є складений ключ

simulation_id, tick, agent_id, оскільки один агент має один snapshot у конкретний tick конкретної симуляції. JSON-поля застосовано для складних структур, які не потребують окремого реляційного розкладання в межах MVP: потреби, persona, schedule, options і метадані.

Логічну модель даних наведено на рисунку 2.12.

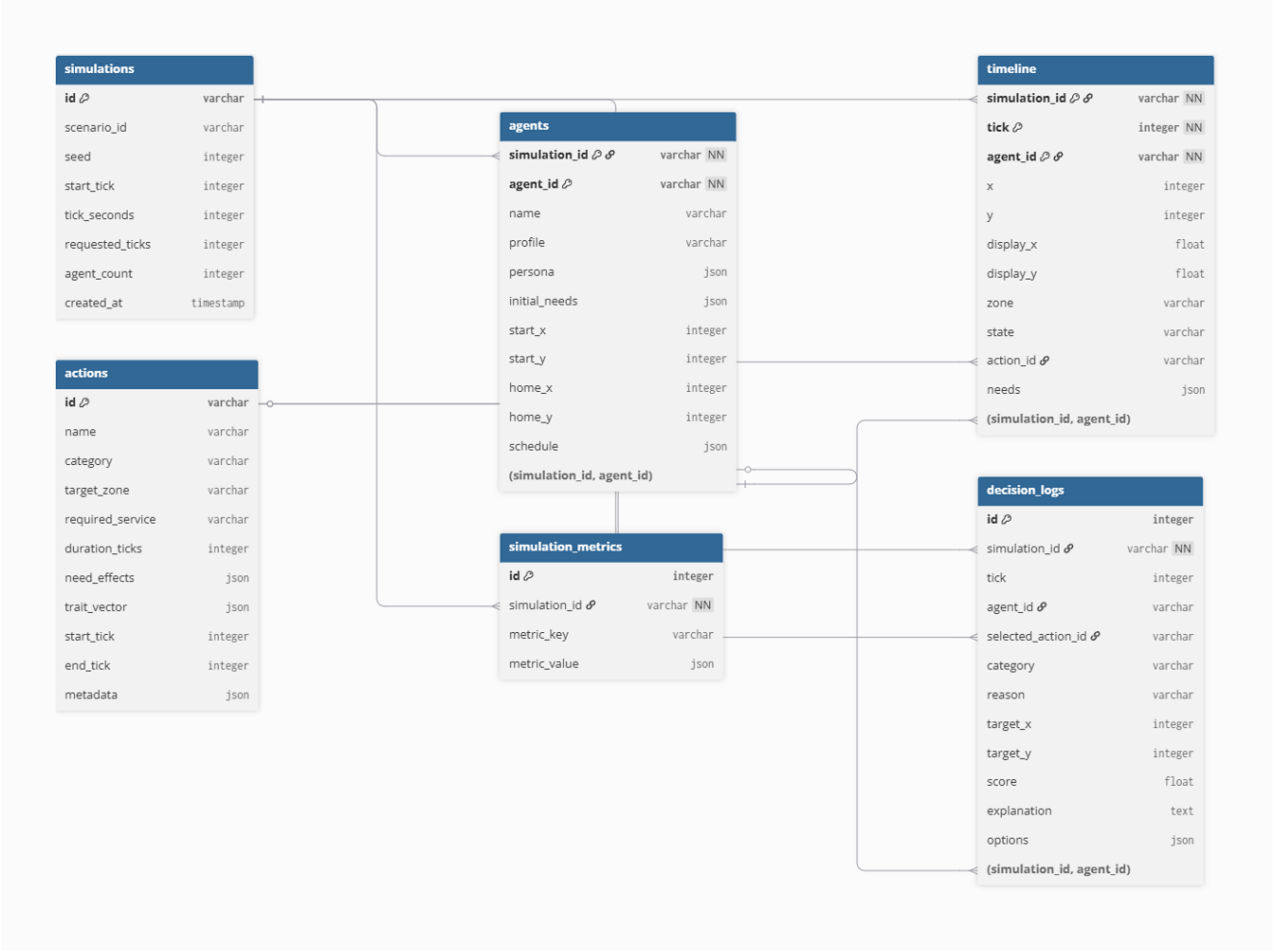


Рисунок 2.12 – Логічна модель даних SQLite

3 РЕАЛІЗАЦІЯ ПРОГРАМНОЇ СИСТЕМИ

У цьому розділі описано архітектуру, доменні моделі, карту, генерацію агентів, математичну модель вибору дій, маршрутизацію, симуляційний рушій, SQLite-сховище, HTTP API, клієнтський інтерфейс і контейнерний запуск. Реалізацію виконано мовою Python, що відповідає можливостям локального запуску без складної серверної інфраструктури [8].

3.1 Загальна архітектура системи

Система побудована за принципом поділу на дані, обчислювальне ядро, шар збереження, HTTP API та клієнтську частину. Дані зберігаються у папці data: карта, дії, архетипи агентів і сценарії. Обчислювальне ядро міститься у пакеті urban_sim. SQLite-сховище реалізує модуль repository. HTTP API обслуговує запити браузера. Клієнтська частина розміщена у папці web і працює як плеєр попередньо згенерованого запису.

Ключове архітектурне рішення поділ на фазу генерації та фазу відтворення. У фазі генерації backend завантажує карту, дії, сценарій, створює агентів, запускає SimulationEngine і записує результат у SQLite. У фазі відтворення браузер отримує готовий recording через API, після чого локально програє кадри, інтерполює рух агентів і відображає деталі.

Такий поділ має практичні переваги. По-перше, браузер не виконує складну симуляційну логіку. По-друге, результат можна ставити на паузу, перемотувати та повторно аналізувати. По-третє, decision logs і metrics уже збережені, тому помилки в поведінці можна діагностувати після завершення генерації. По-четверте, одна симуляція стає відтворюваним експериментальним артефактом.

Основні модулі реалізації:

– модуль urban_sim/models.py містить доменні сутності Needs, Persona, ScheduledTask, Action, Agent, AgentSnapshot, TimelineFrame, DecisionLog і Recording;

– модуль urban_sim/map_loader.py відповідає за завантаження карти, опис Tile і CityMap, читання Tiled-подібних шарів, entrances і сервісів;

– модуль urban_sim/agent_factory.py створює населення агентів, розподіляє домашні позиції, застосовує jitter до потреб, persona і розкладу;

– модуль urban_sim/action_selector.py реалізує вибір дій через ActionSelector, cosine similarity, needScore, costScore, utility та softmax;

– модуль urban_sim/pathfinding.py містить A*-маршрутизацію;

– модуль urban_sim/engine.py формує timeline і decision logs;

– модуль urban_sim/repository.py зберігає й читає симуляції зі SQLite;

– модуль urban_sim/api.py обслуговує HTTP-запити;

– модуль urban_sim/cli.py забезпечує локальні команди init-db, generate і serve;

– клієнтський файл web/app.js відповідає за візуалізацію;

– tools/export_diploma_charts.py формує CSV-дані для діаграм.

3.2 Модель даних предметної області

У Python-коді доменні сутності реалізовано як dataclass-структури. Використання dataclasses дає змогу компактно описувати поля сутностей, автоматично отримувати ініціалізацію та зберігати зрозумілу структуру об’єктів [9]. У системі це особливо важливо, тому що одні й ті самі сутності проходять через кілька шарів: JSON-файли, рушій симуляції, SQLite, API та браузер.

Клас Needs зберігає вісім потреб: energy, hunger, happiness, sociality, health, money, stress, productivity. Метод decayed повертає новий об’єкт Needs з урахуванням природної зміни потреб залежно від стану агента. Для idle, moving, working і acting використовуються різні decay rates. Метод apply_effects додає ефекти завершеної дії до потреб і обмежує результат допустимим діапазоном від нуля до одиниці.

Клас Persona містить словник рис агента. У реалізації використано economy, social, activity і diligence. Метод vector_for повертає числовий вектор у порядку

заданих ключів. Це потрібно, щоб порівнювати persona агента з trait_vector дії через косинусну подібність.

Клас ScheduledTask описує обов'язкову або пріоритетну задачу. Він має action_id, start_tick і end_tick. Метод active_at перевіряє, чи поточна модельна хвилина потрапляє у часовий інтервал. Це дозволяє задати робочі, навчальні, харчові або рекреаційні вікна без жорсткого кодування в рушії.

Клас Action описує одну доступну дію. Поле category визначає поведінкову категорію. target_zone вказує, до якої зони потрібно прямувати. required_service уточнює, що агент має шукати не просто зону, а конкретний сервіс, наприклад food, shopping, study, hospital, leisure або home_food. duration_ticks задає тривалість дії. У поточній реалізації дії мають реалістичніші тривалості: короткий перекус триває приблизно 15 tick, прийом їжі 30-35 tick, робота 45 tick, навчання 40 tick, відпочинок удома 50 tick. Це зменшує смикання агентів між діями. need_effects описує зміну потреб після завершення. trait_vector описує, яким persona-рисам дія відповідає. start_tick і end_tick задають часову доступність. metadata використовується для ознак disabled, compulsory, fallback, quick або сценарних властивостей.

Клас Agent містить id, name, persona, needs, position, home_position і schedule. Поле position є поточною маршрутною позицією для старту. Поле home_position використовується для дій у житловій зоні, щоб агенти поверталися до свого призначеного житла, а не до найближчого випадкового дому. Метод scheduled_action_id повертає дію з активного розкладу.

Класи AgentSnapshot, TimelineFrame, DecisionLog і Recording описують результат симуляції. AgentSnapshot зберігає стан одного агента в одному кадрі. TimelineFrame містить tick і список snapshot усіх агентів. DecisionLog фіксує обране рішення та альтернативи. Recording є повним записом симуляції з simulation_id, tick_seconds, frames, decision_logs і metrics.

Базове оновлення потреб задається формулою:

$$N_i(t+1) = clip\big(N_i(t) + D\big(s_i(t)\big) + E\big(u_i(t)\big), 0{,}1\big) \tag{3.1}$$

де $N_i(t)$ – вектор потреб агента i у момент часу t;

$clip$ – обмеження кожного значення діапазоном від нуля до одиниці;

$D(s_i(t))$ – зміна потреб залежно від стану idle, moving, working або acting;

$E(u_i(t))$ – ефект завершеної дії.

У практичній реалізації D застосовується кожного tick, а E додається після завершення дії. Саме тому величини decay повинні бути малими, інакше агент буде швидко переходити до критичних станів.

3.3 Модель карти міського середовища

Карта міського середовища зберігається у форматі Tiled JSON. Використання Tiled-подібної структури дає змогу описувати тайли, шари, властивості та object layers у зручному для редагування вигляді [10]. У поточній реалізації карта має розмір 60 на 38 тайлів. До функціональних зон належать residential, commercial, work, recreational, healthcare і education. До інфраструктурних зон належать sidewalk, road, crosswalk і ground.

Кожен Tile має поля zone, walkable, services, label, building_id, road_type, entrance_for і travel_cost. Поле zone визначає семантичний тип. Поле walkable визначає, чи може агент використовувати тайл у маршруті. Поле services містить сервіси, доступні на тайлі. Поле building_id дозволяє групувати входи та клітинки однієї будівлі. Поле entrance_for показує, входом до яких зон є тайл. Поле travel_cost використовується алгоритмом маршрутизації.

У системі реалізовано важливе правило: функціональні зони можуть бути непрохідними для маршрутизації, але доступними для виконання дій через входи. Це означає, що агент не прокладає маршрут крізь будівлю. Він іде до прохідного entrance tile, який розташований на тротуарі або на межі зони, а потім переходить у режим внутрішнього перебування. Така логіка запобігає руху крізь стіни.

Карта має прямокутну структуру кварталів: функціональна зона, навколо неї одинарний тротуар, навколо тротуару подвійна дорога. Пішохідні переходи з’єднують тротуари через дорогу. Ground використовується як декоративна межа

або фон і не повинен утворювати порожні смуги біля зон. У фінальній візуальній концепції місто має виглядати як упорядкований район з прямокутними кварталами, зрозумілими зонами, дорогами та переходами.

Семантичну структуру карти наведено на рисунку 3.3.

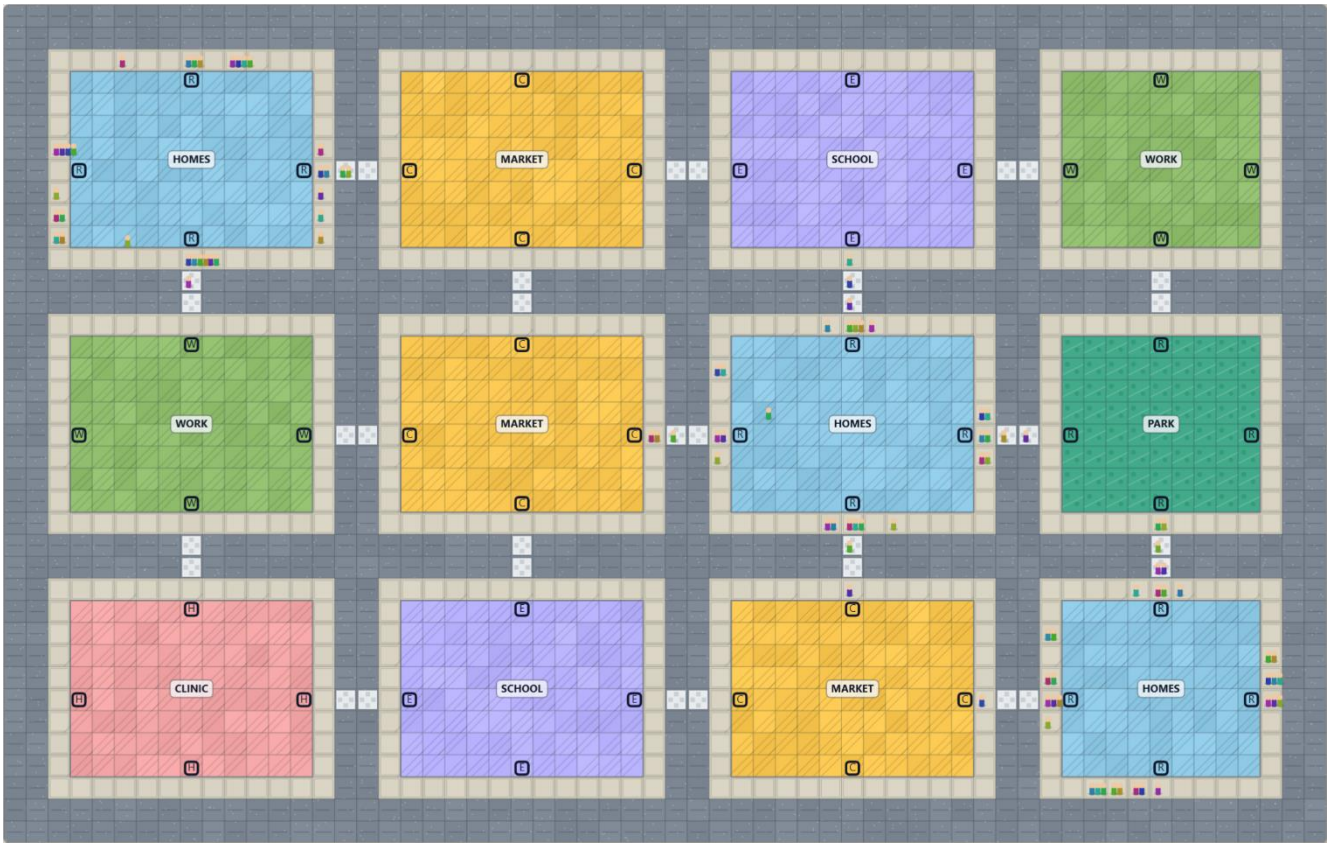


Рисунок 3.3 – Семантична структура тайлової карти міського середовища

3.4 Генерація населення агентів

Генерація агентів реалізована у модулі agent_factory. Вхідними даними є city_map, список архетипів, кількість агентів, seed і start_tick. Кількість агентів обмежується функцією clamp_agent_count, щоб користувач не міг випадково створити надто малу або надто велику популяцію. Поточний діапазон становить від 10 до 200 агентів.

Спочатку генератор шукає житлові позиції. Для цього він проходить усі тайли карти й знаходить ті, що мають entrance_for residential або сервіс home_food. Позиції групуються за building_id, щоб агенти розподілялися між різними

житловими будівлями. Потім список building_id перемішується seed-контрольованим генератором випадкових чисел. Для кожного агента домашня позиція вибирається циклічно по будинках і входах, що запобігає ситуації, коли всі агенти живуть в одній зоні.

Після домашньої позиції вибирається стартова позиція. Система шукає прохідні тайли поблизу дому в межах мангеттенської відстані до двох клітинок. Це дозволяє агентам починати симуляцію біля свого житла, але не обов’язково на одному тайлі. Якщо житлових стартових позицій немає, використовується загальний список прохідних тайлів.

Далі агент отримує ім’я, persona, needs і schedule. Імена вибираються з базового списку з числовим індексом. Persona і needs не копіюються дослівно з архетипу, а змінюються через jitter. Для persona додається випадкове відхилення приблизно в межах 0.18, для потреб – приблизно в межах 0.16. Після цього потреби стабілізуються відповідно до start_tick. Якщо симуляція починається у робочий час, агент не повинен стартувати з критично низькою енергією або ситістю; тому застосовуються мінімальні пороги для energy, hunger, happiness, sociality, health і productivity та верхня межа для stress. Це робить старт симуляції правдоподібнішим.

Розклад також варіюється. Кожна запланована дія зсувається на випадкові 15-хвилинні кроки в межах приблизно 45 хвилин, а тривалість може змінюватися в межах 30 хвилин. Це робить агентів різними навіть тоді, коли вони створені з однакових архетипів. У результаті частина агентів має більше робочих блоків, частина – навчальних, частина – медичних або рекреаційних, що зменшує одноманітність поведінки.

Процес генерації населення наведено на рисунку 3.4.

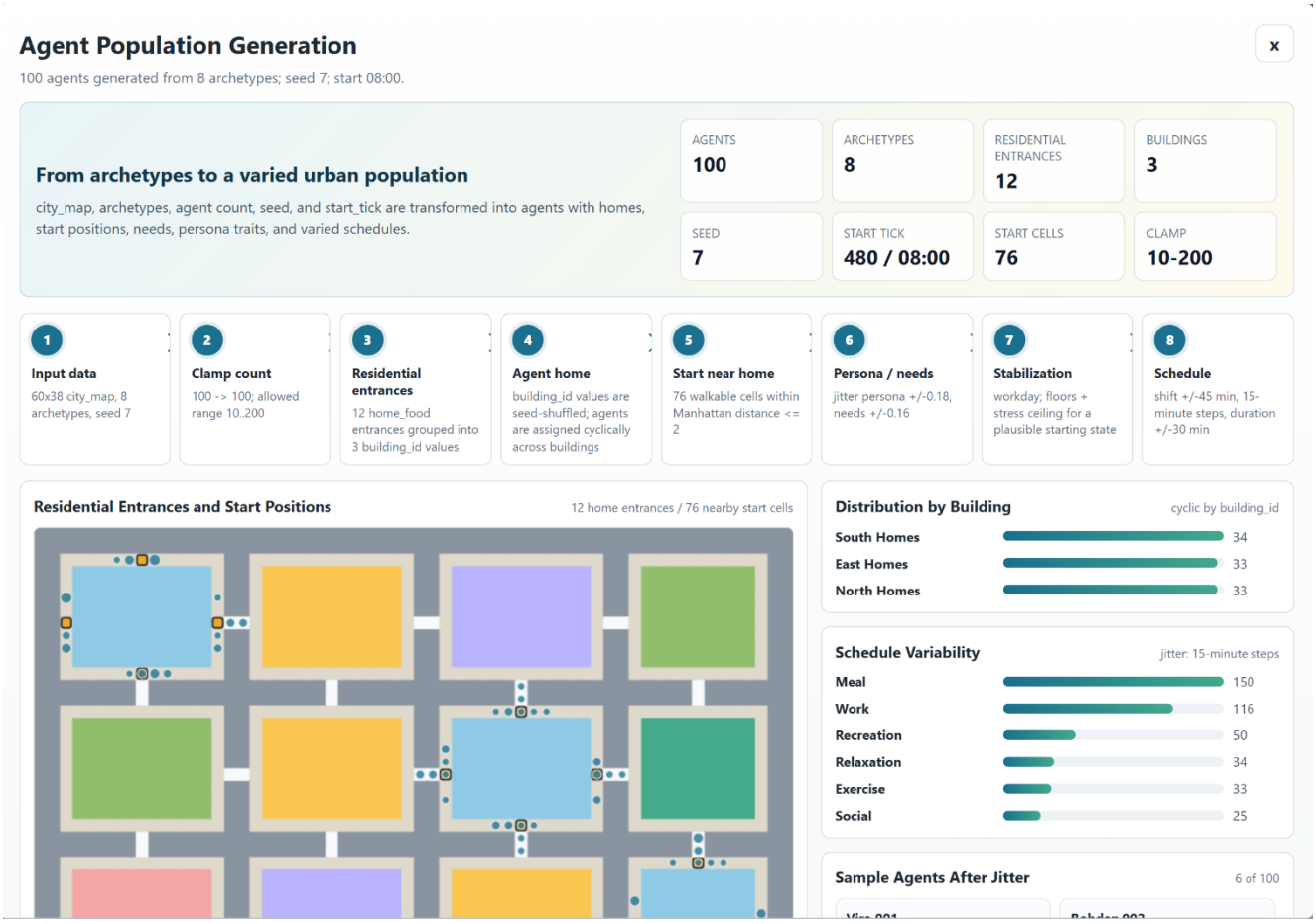


Рисунок 3.4 – Генерація населення агентів на основі архетипів і житлових входів

3.5 Математична модель вибору дії

Вибір дії реалізовано у класі ActionSelector. Модель поєднує правила, розклад, потреби, similarity між агентом і дією, cost дії та softmax-вибір. Її мета – зробити поведінку не повністю випадковою, але й не абсолютно детермінованою. Агент повинен мати схильність до логічних дій, але різні агенти в однаковій ситуації не повинні поводитися ідентично.

Спочатку визначається категорія поведінки. Якщо hunger менше critical_hunger, агент переходить у режим critical_hunger і шукає їжу. Поріг critical_hunger зменшено до 0.18, щоб агент не кидав роботу або іншу дію занадто рано. Якщо активний ScheduledTask, запланована дія має пріоритет. Якщо критичного стану і запланованої дії немає, система враховує час доби: meal window, work window, evening window і night window. Наприклад, у робоче вікно агент з

достатньою енергією, ситістю і здоров'ям може вибрати work. У вечірнє вікно при високому stress перевага надається stress relief, при низькій sociality - social, при низькому happiness - recreation.

Для оцінки дії формується об'єднання ключів persona і trait_vector дії. Косинусна подібність обчислюється так:

$$sim(P, A) = \frac{\sum_{k=1}^{m} P_k A_k}{\sqrt{\sum_{k=1}^{m} P_k^2} \sqrt{\sum_{k=1}^{m} A_k^2}} \tag{3.2}$$

де P – вектор рис агента;

A – вектор рис дії;

m – кількість ознак.

Якщо один із векторів має нульову довжину, similarity дорівнює нулю.

Оцінка користі для потреб визначається через позитивний ефект дії та дефіцит відповідної потреби:

$$needScore = min\left(1, \sum_{j} max(0, e_j)(1 - N_j) + max(0, -e_{stress})stress\right) \tag{3.3}$$

де e_j – ефект дії для потреби j;

N_j – поточний рівень потреби;

$stress$ – поточний рівень стресу агента.

Оцінка вартості враховує витрати money, health, energy і збільшення stress. Підсумкова utility-оцінка дії:

$$U(u) = 0.55 \cdot sim(P, A) + 0.40 \cdot needScore(u) - 0.12 \cdot costScore(u) \tag{3.4}$$

де $costScore(u)$ – нормалізована вартість дії.

Вага 0.55 робить persona-сумісність найважливішою складовою. Вага 0.40 зберігає сильний вплив актуальних потреб. Віднімання 0.12 зменшує привабливість надто дорогих дій.

Після ранжування система бере top-k дій і перетворює utility на імовірності:

$$Pr(u_i) = \frac{exp\big((U_i - U_{max})/\tau\big)}{\sum_{r=1}^{k} exp\big((U_r - U_{max})/\tau\big)} \tag{3.5}$$

де U_{max} – найбільша utility серед top-k;

τ – temperature;

k – кількість альтернатив.

Віднімання U_{max} підвищує числову стабільність softmax. Саме softmax забезпечує різноманітність: найкраща дія має більшу ймовірність, але інші достатньо близькі альтернативи також можуть бути обрані.

У лістингу 3.1 показано, як програмна реалізація поєднує відповідність persona, користь для поточних потреб і вартість дії. У повному методі до цієї оцінки додатково застосовуються часові обмеження, пріоритети розкладу, категорія поведінки та softmax-вибір із набору найкращих альтернатив.

Лістинг 3.1 – Фрагмент обчислення utility-оцінки дії

```
trait_keys = sorted(set(agent.persona.traits) |
set(action.trait_vector))
similarity = cosine_similarity(
    agent.persona.vector_for(trait_keys),
    [action.trait_vector.get(key, 0.0) for key in trait_keys],
)
need_score = self._need_score(agent.needs, action.need_effects)
cost_score = self._cost_score(action, city_map, agent.position)
utility = 0.45 * need_score + 0.30 * similarity - 0.15 * cost_score
```

Повну послідовність вибору дії наведено на рисунку 3.5.

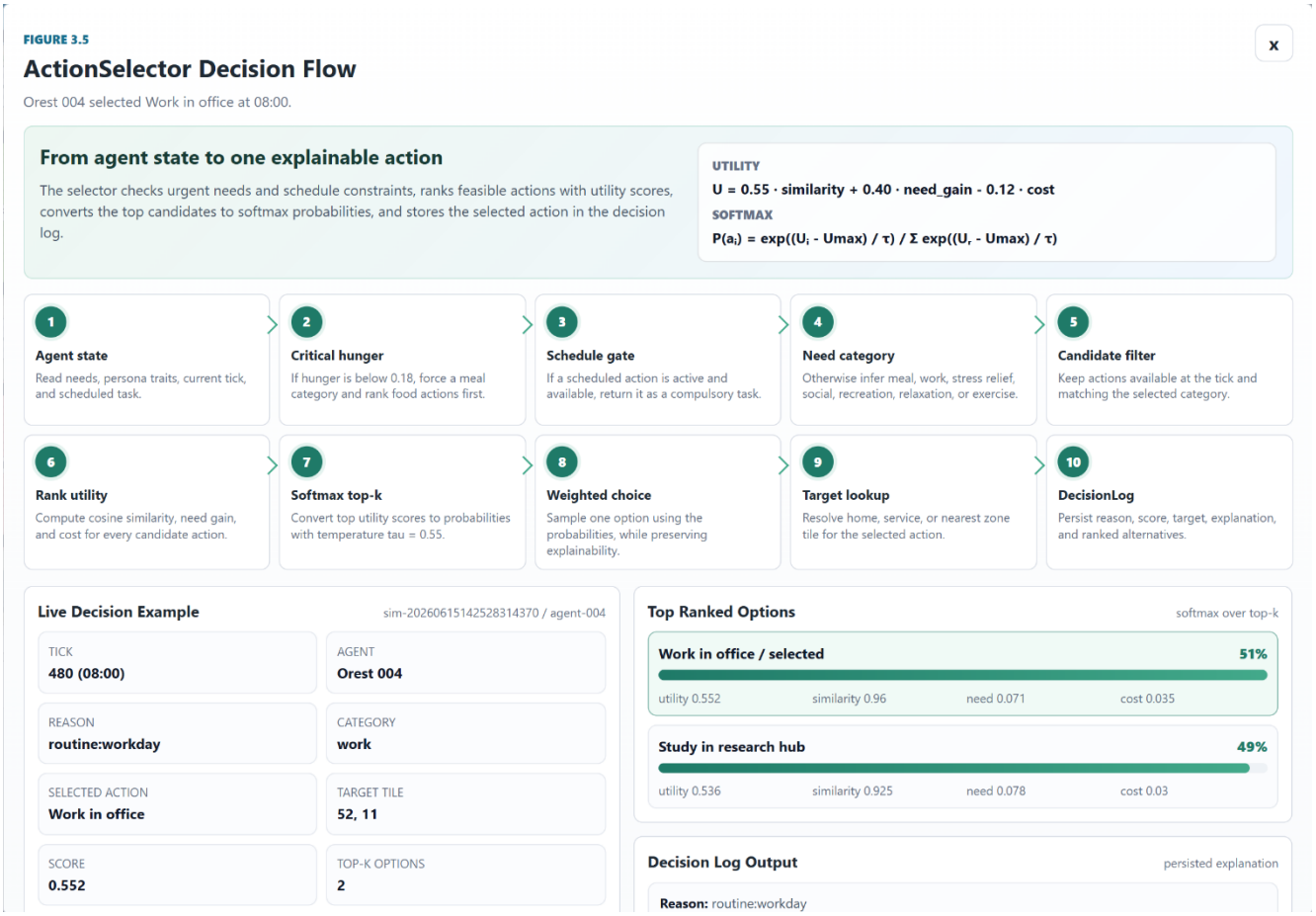


Рисунок 3.5 – Послідовність вибору дії агентом у модулі ActionSelector

3.6 Алгоритм маршрутизації та логіка входу до зон

Маршрутизацію реалізовано через A*. Карта інтерпретується як граф, у якому вершинами є прохідні тайли. Сусідами є чотири ортогональні клітинки: праворуч, ліворуч, униз і вгору. Якщо тайл непрохідний, він не потрапляє до списку сусідів. Це означає, що road, ground і внутрішні клітинки будівель не використовуються для маршруту агента.

Пріоритет вузла в A* обчислюється формулою:

$$f(n) = g(n) + h(n) \tag{3.6}$$

де $g(n)$ – накопичена вартість шляху до вузла;

$h(n)$ – евристика.

Оскільки рух сітковий, евристикою є мангеттенська відстань:

$$h(n) = |x_n - x_t| + |y_n - y_t| \tag{3.7}$$

де (x_t, y_t) – цільова координата. Якщо шлях не існує, функція astar_path генерує NoPathError. Рушій симуляції в такому випадку залишає агента idle, а не змушує його проходити через заборонені тайли.

Лістинг 3.2 демонструє, що маршрути будуються тільки через сусідів, які дозволені картою. Отже, агент не може перетинати непрохідні тайли, дороги або внутрішні частини будівель, а переміщення до функціональних зон відбувається через визначені входи.

Лістинг 3.2 – Основний цикл A*-маршрутизації

```
    while frontier:
        _, current = heapq.heappop(frontier)
        if current == goal:
            break
        for next_pos in city_map.neighbors(current):
            new_cost = cost_so_far[current] +
city_map.travel_cost(next_pos)
            if next_pos not in cost_so_far or new_cost <
cost_so_far[next_pos]:
                cost_so_far[next_pos] = new_cost
                priority = new_cost + manhattan(next_pos, goal)
                heapq.heappush(frontier, (priority, next_pos))
                came_from[next_pos] = current
```

Для дій у функціональних зонах ціль визначається не випадковою клітинкою всередині будівлі, а найближчим entrance або service tile. Якщо дія має required_service, карта шукає nearest_service_tile. Якщо сервіс не заданий, шукається nearest_zone_tile, який може враховувати entrance_for. Для residential і home_food використовується assigned home_position агента, щоб він повертався до власного житла.

Після досягнення входу запускається внутрішній режим. Метод _zone_tiles_for_entry знаходить компоненту зони, яка торкається конкретного входу. Це важливо, якщо на карті є кілька зон одного типу: агент, який зайшов через

один вхід, не повинен телепортуватися у віддалену будівлю того самого типу. Усередині зони агент отримує display_position з дробовими координатами та indoor_target. Метод _advance_indoor_motion плавно пересуває агента всередині зони, але цей рух не вважається маршрутом між зонами і не повинен малюватися як червона зовнішня траєкторія.

Агенти в одній зоні не повинні накладатися один на одного, якщо є вільний простір. Для цього _indoor_point генерує кілька кандидатів і вибирає той, який має найбільшу мінімальну відстань до вже зайнятих indoor-позицій у цій зоні. Це не є фізичною симуляцією зіткнень, але достатньо для MVP, щоб агенти візуально розміщувалися природніше.

3.7 Рушій симуляції та формування timeline

Рушій симуляції реалізовано у класі SimulationEngine. Він отримує city_map, actions, agents, tick_seconds, seed, start_tick і selector. Під час ініціалізації створюється random.Random із seed, кеш зон, кеш компонент входу та runtime-об’єкти агентів. Runtime-об’єкт містить копію агента, маршрутну позицію, display_position, шлях, поточну дію, кількість tick до завершення дії, стан, indoor_zone, indoor_target і список indoor_tiles.

Метод run(max_ticks) виконує симуляцію для заданої кількості tick. На кожному tick він викликає _step_agent для кожного runtime-агента, збирає AgentSnapshot і формує TimelineFrame. Після завершення всіх tick рушій обчислює metrics і додає agent_journals. Результатом є Recording.

Метод _step_agent виконує основну логіку. Спочатку потреби агента оновлюються через decayed. Якщо агент виконує дію, зменшується action_ticks_left, стан стає acting, виконується внутрішній рух у зоні, а після завершення дії застосовуються need_effects. Якщо агент має шлях, він переходить на наступний тайл, position оновлюється, стан стає moving. Якщо дії і шляху немає, викликається ActionSelector, рішення записується у decision_logs, маршрут будується через

astar_path, після чого агент або починає рух, або одразу запускає дію, якщо вже перебуває на цілі.

Метод _start_action переводить агента в режим acting. Він встановлює action_ticks_left, indoor_zone, indoor_tiles, display_position і indoor_target. Якщо дія триває один tick, її ефекти застосовуються одразу. У поточних налаштуваннях більшість дій триває довше, тому агент залишається всередині зони достатній час і не смикається між потребами кожні кілька хвилин.

Логіку одного tick наведено на рисунку 3.6.

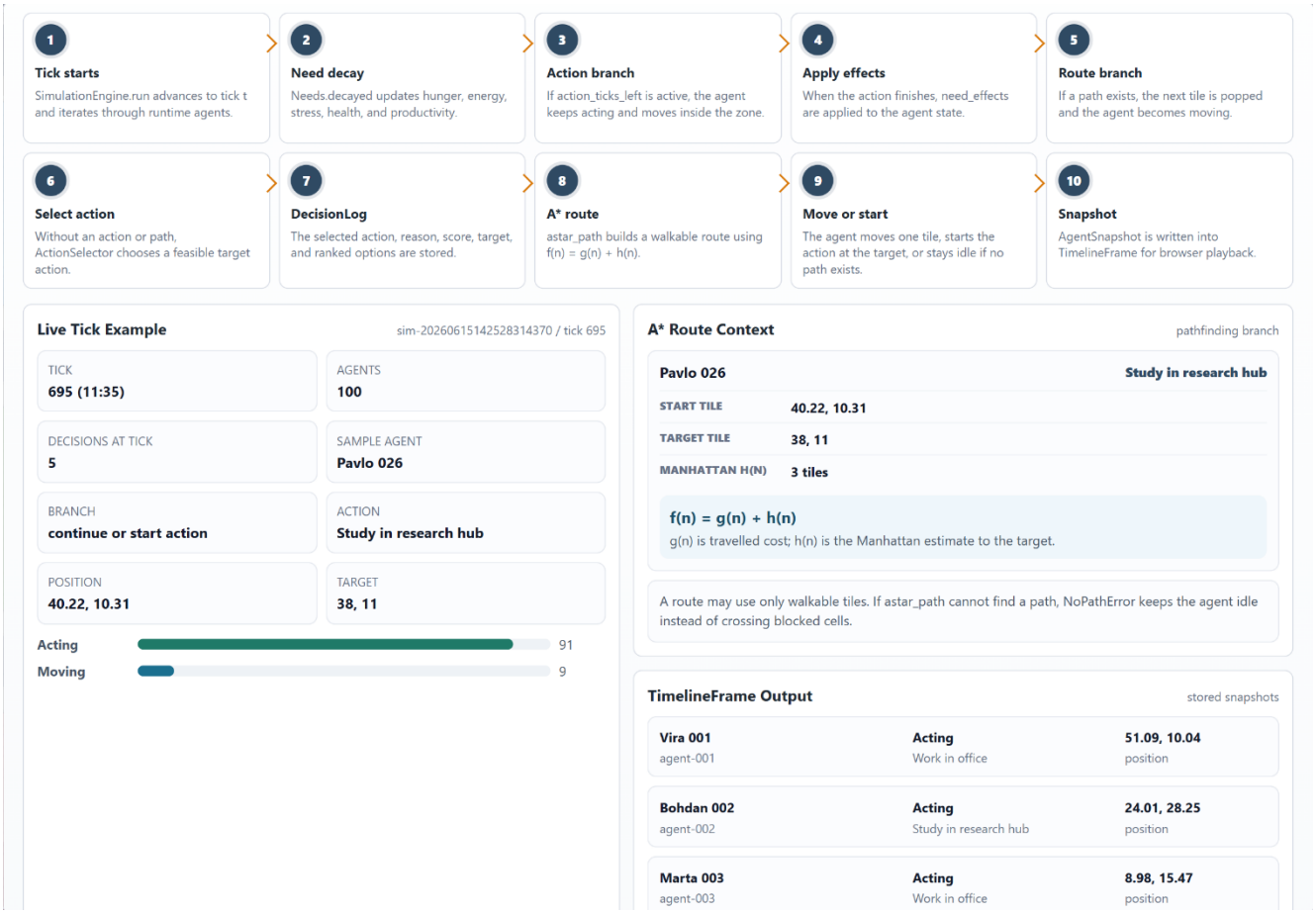


Рисунок 3.6 – Робота SimulationEngine на одному tick

3.8 SQLite-сховище, журнали та метрики

Для збереження використано SQLite. Вбудована база даних підходить для локального прототипу, оскільки не потребує окремого сервера, легко переноситься разом із проєктом і підтримує SQL-запити для аналізу результатів [11].

Сховище містить логічні групи даних. Actions зберігають action space, щоб система могла відновити назви й параметри дій. Agents зберігають архетипи. Simulations зберігають ідентифікатор симуляції, дату створення та tick_seconds. Timeline зберігає стан кожного агента в кожному tick: координати, needs, action_id і state. Decision logs зберігають рішення агентів, причини вибору, цілі, оцінки й альтернативи. Simulation metrics зберігають агреговані результати у JSON.

Метрики будуються після завершення run. average_needs обчислює середнє значення кожної потреби по всіх snapshot. action_counts рахує кількість snapshot з певним action_id. zone_visits рахує, у яких зонах перебували агенти. state_counts рахує moving, acting, idle та інші стани. total_frames зберігає кількість кадрів, total_agent_samples – кількість зразків агентів.

Decision logs є основою пояснюваності. Кожний запис містить не тільки selected_action_id, а й reason, score, target і options. У options зберігаються probability, utility_score, similarity_score, need_score і cost_score для альтернатив. Тому користувач може побачити, що агент вибрав роботу через розклад, їжу через критичний голод, парк через стрес або соціальну дію через низьку sociality.

3.9 HTTP API, CLI та керування симуляціями

HTTP API реалізовано у модулі api. Сервер віддає статичні файли web і JSON-ендпоїнти. GET /api/map повертає карту. GET /api/actions повертає простір дій. GET /api/agents повертає архетипи агентів. GET /api/scenarios повертає доступні сценарії. GET /api/simulations повертає список збережених симуляцій. GET /api/simulations/latest повертає останній запис або створює початкову симуляцію, якщо записів немає. GET /api/simulations/{id} повертає конкретний Recording.

POST /api/simulations/generate створює нову симуляцію. DELETE /api/simulations/{id} видаляє одну симуляцію. DELETE /api/simulations видаляє всі записи.

Під час генерації API читає JSON-body, нормалізує ticks, start_tick, seed, agent_count і scenario_id. Кількість агентів обмежується clamp_agent_count. Тривалість не повинна виходити за межі допустимого діапазону. Якщо scenario_id заданий, сценарій застосовується до карти та дій. Потім запускається generate_agents із передачею start_tick, щоб стартові потреби агентів відповідали фазі дня, і SimulationEngine.run. Після цього Recording записується в SQLite, а frontend отримує відповідь зі статусом готовності.

Лістинг 3.3 ілюструє роль API як координаційного шару. Обробник не містить правил поведінки агента і не виконує візуалізацію, а приймає параметри, запускає потрібні сервіси, зберігає результат і повертає клієнту ідентифікатор готової симуляції.

Лістинг 3.3 – Фрагмент обробки запиту генерації симуляції

```
    payload = json.loads(self.rfile.read(content_length) or b"{}")
    request = SimulationRequest.from_payload(payload)
    city_map = load_city_map(request.map_path)
    actions = load_actions(request.actions_path)
    agents = generate_agents(city_map, request.agent_count,
request.seed, request.start_tick)
    recording = SimulationEngine(city_map, actions, agents,
request.seed).run(
        ticks=request.ticks,
        start_tick=request.start_tick,
    )
    repository.save_recording(recording)
    self._send_json({"simulation_id": recording.simulation_id,
"status": "ready"})
```

CLI реалізовано у модулі cli. Команда init-db створює схему SQLite та записує початкові actions і agents. Команда generate створює симуляцію з параметрами ticks, start-tick, seed і agent-count. Команда serve запускає HTTP-сервер. Така структура дозволяє використовувати систему як із браузера, так і з командного рядка.

У реалізації статичного сервера враховано базову безпеку: шлях до файлу нормалізується через resolve, після чого перевіряється, що цільовий файл не виходить за межі web_dir. Такий захист потрібний для запобігання path traversal при обслуговуванні локальних статичних файлів; загальні принципи безпечного тестування вебзастосунків описані в OWASP Web Security Testing Guide [12].

3.10 Клієнтська частина та візуалізація

Клієнтська частина складається з index.html, styles.css і app.js. Візуалізація карти реалізована через Canvas API, який дає змогу малювати тайли, текстури, агентів, маршрути, heatmap і overlay-елементи у 2D-контексті [13].

Дані з API завантажуються через Fetch API. Це дає змогу отримувати карту, actions, scenarios, simulations і recording без перезавантаження сторінки [14].

Модальне вікно генерації побудовано на HTML dialog element. Воно містить параметри start time, duration, agent count, seed і scenario, а після успішної генерації повідомляє, що симуляція готова до програвання [15].

У app.js зберігається глобальний state. Він містить cityMap, actions, scenarios, simulations, recording, currentIndex, playing, speed, selectedAgentId, selectedTile, showHeatmap, showGrid, camera і pointer. Camera має scale, minScale, offsetX і offsetY. Pointer зберігає стан drag-перетягування. Завдяки цьому UI може плавно масштабувати й переміщувати карту незалежно від даних симуляції.

Функція initialize завантажує карту, дії, сценарії, список симуляцій і останній recording. Після цього вона заповнює селектори, рендерить легенду, підлаштовує canvas і запускає первинний render. Функція resizeCanvas враховує devicePixelRatio, CSS-розміри canvas і розмір карти. Мінімальний масштаб розраховується так, щоб у крайньому віддаленні користувач бачив усе місто без великих порожніх полів.

Масштабування реалізовано функцією zoomAt. Перед зміною масштабу обчислюється world coordinate під курсором. Після зміни scale offset перераховується так, щоб та сама точка залишалась під курсором. Це робить zoom колесом миші природним. Перетягування реалізовано через mousedown,

mousemove і mouseup. Якщо користувач рухав карту, наступний mouseup не сприймається як клік по агенту або зоні.

Функція render є центральною. Вона викликає drawMap, drawHeatmap, drawZoneOverlays, drawEntrances, drawSelectedTrail, drawAgents і оновлення HTML-панелей. Порядок важливий: карта малюється першою, потім heatmap, потім контури зон, входи, маршрут і агенти. Агент відображається як стилізований гліф із кольором, отриманим із hash agent_id. Вибраний агент виділяється червоним.

Права глобальна панель інтерфейсу містить тільки загальні блоки Metrics, Needs History і Legend. Деталі агента або зони винесені в offcanvas. Якщо користувач вибирає агента, offcanvas показує Current state, Decision, Schedule, Decision journal і Route around current tick. Якщо користувач вибирає зону, offcanvas показує Zone state, Doors, Agents in zone і Available actions. Журнали мають власні прокручувані області, стабільні розміри, touch-friendly scroll і коректне відображення довгих пояснень.

Плавне відтворення реалізовано через requestAnimationFrame, що є стандартним механізмом синхронізації браузерної анімації з частотою оновлення екрана [16]. Backend зберігає дискретні кадри, але frontend інтерполює координати між поточним і наступним кадром:

$$p(\alpha) = p_t + (p_{t+1} - p_t) \tag{3.8}$$

де p_t – позиція агента у поточному кадрі;

p_{t+1} – позиція у наступному кадрі;

α – прогрес анімації від нуля до одиниці. Це робить рух візуально плавним без збільшення кількості записів у SQLite.

Карта, інтерфейс і offcanvas-панель наведені на рисунку 3.8.

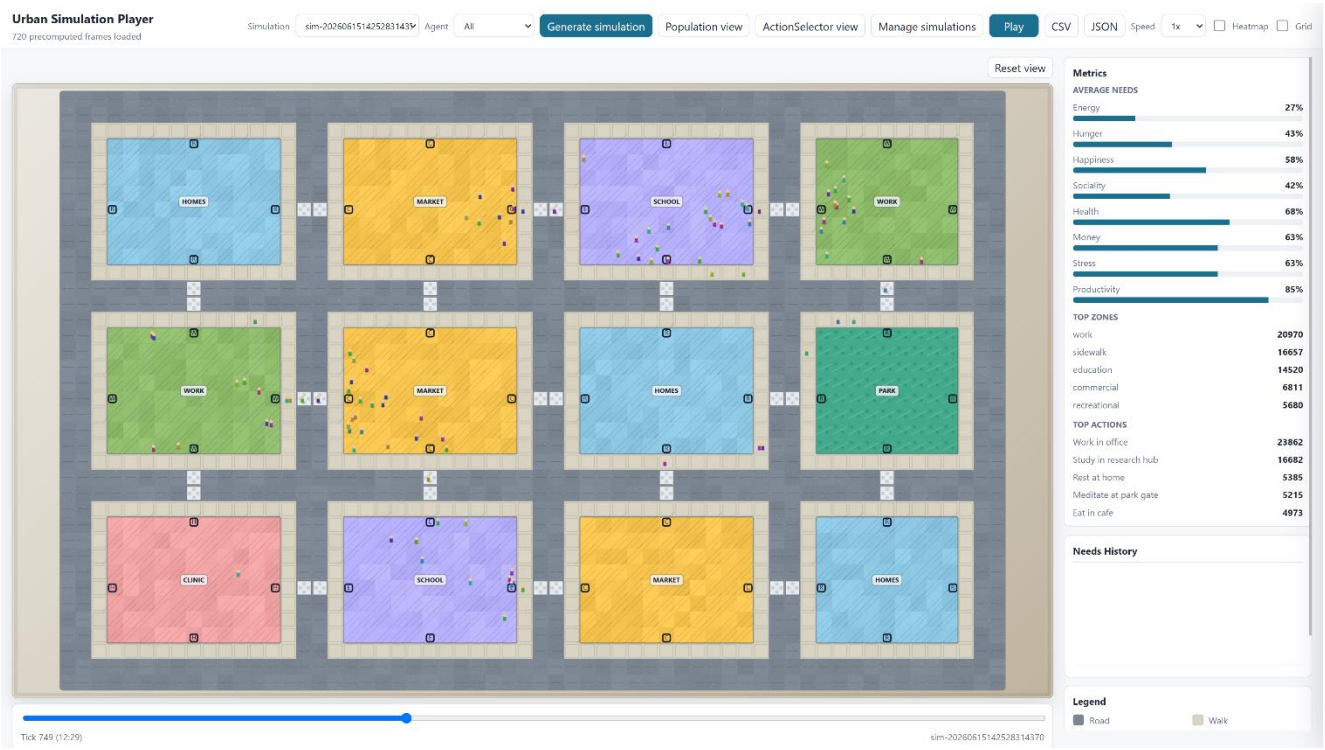


Рисунок 3.8 – Клієнтський інтерфейс вебплеєра симуляції

З погляду вебстандартів структура сторінки спирається на HTML-документ [17]. Дані карти, дій, агентів, сценаріїв і симуляцій передаються як JSON, тобто у форматі, визначеному ECMA-404 [18].

3.11 Контейнеризація і структура репозиторію

Вихідний код системи опубліковано в публічному репозиторії GitHub (див. додаток Ж). Репозиторій містить backend, frontend, карту, сценарії, тести, Docker-конфігурацію, документацію та допоміжні інструменти для експорту аналітичних даних.

Для спрощення запуску систему контейнеризовано. Docker дозволяє запускати застосунок в ізольованому середовищі з однаковими залежностями [19].

Docker Compose використовується для опису сервісу, порту, змінних середовища та volume для SQLite-бази [20]. Типова команда запуску: docker compose up --build. Після запуску вебінтерфейс доступний за адресою http://127.0.0.1:8000/.

У контейнерному режимі SQLite-файл зберігається у Docker volume. Це означає, що симуляції не зникають після перезапуску контейнера. Додатково підтримуються змінні SIM_AUTO_GENERATE, SIM_GENERATE_TICKS, SIM_GENERATE_START_TICK, SIM_GENERATE_SEED і SIM_GENERATE_AGENT_COUNT. Вони дозволяють автоматично створити симуляцію при старті.

Структура репозиторію організована так, щоб дані, backend, frontend, тести, документи й інструменти були розділені. У папці data зберігаються карта, дії, архетипи та сценарії. У папці urban_sim розміщено backend-логіку. У папці web розташовано статичний клієнт. У папці tools є допоміжні скрипти, зокрема експорт даних для діаграм. CSV-файли результатів генеруються локально і не є обов'язковою частиною репозиторію. Така структура спрощує підтримку, тому що зміна карти або action space не потребує переписування рушія симуляції.

4 ТЕСТУВАННЯ, ВПРОВАДЖЕННЯ ТА АНАЛІЗ РЕЗУЛЬТАТІВ

У цьому розділі описано перевірку працездатності програмної системи, коректності карти, маршрутів, поведінки агентів, збереження даних, браузерної взаємодії, продуктивності та аналітичних результатів. Тестування розглянуто як невід'ємну частину життєвого циклу розробки, оскільки якість симуляційної системи визначається не тільки відсутністю помилок у коді, а й відповідністю поведінки агентів, карти та метрик поставленим вимогам [21].

4.1 Стратегія та види тестування

Для перевірки системи використано кілька взаємодоповнювальних видів тестування. Модульне тестування перевіряє окремі функції та класи: оновлення потреб, вибір дій, маршрутизацію, завантаження карти, генерацію агентів і роботу репозиторію. Інтеграційне тестування перевіряє взаємодію API, рушія симуляції та SQLite-сховища. Функціональне тестування підтверджує, що користувач може створити симуляцію, відкрити готовий запис, програти timeline, вибрати агента або зону, переглянути offcanvas і видалити симуляцію. Регресійне тестування виконується через фіксовані seed і параметри запуску.

Окремо було передбачено візуальне тестування браузерного інтерфейсу. Воно охоплює перевірку масштабування, перетягування карти, відображення зон, агентів, входів, маршрутів, heatmap і панелей даних. Для продуктивності використовуються контрольні симуляції на 100 агентів і 720-1440 tick, а також спостереження за швидкістю відкриття запису, плавністю Canvas-анімації, обсягом SQLite-файлу та стабільністю прокручування журналів в offcanvas. Для розгортання перевірявся запуск через Docker Compose.

Результат автоматизованого тестування наведено на рисунку 4.1.

```
PS C:\...\urban-agent-sim> python -m unittest discover -s tests
.....................................................
----------------------------------------------------------------------
Ran 61 tests in 0.941s

OK
PS C:\...\urban-agent-sim>
```

Рисунок 4.1 – Результат виконання автоматизованих тестів

4.2 Функціональне та інтеграційне тестування

Функціональна перевірка карти підтвердила, що агенти рухаються тільки прохідними тайлами: тротуарами та пішохідними переходами. Дороги, ground і внутрішні клітинки будівель не використовуються як маршрутні вузли. Входи до зон перевіряються через entrance tiles, які торкаються відповідних функціональних зон. Це забезпечує логіку входу до будівель і запобігає проходженню агентів крізь стіни.

Інтеграційне тестування генерації симуляції перевіряє повний ланцюг від HTTP-запиту до готового запису в SQLite. Після POST-запиту система нормалізує параметри, завантажує карту, дії та сценарій, генерує агентів, запускає рушій, формує timeline, decision logs і metrics, зберігає Recording та повертає клієнту статус готовності. Для перевірки видалення симуляцій використовуються DELETE-запити для одного запису і для всієї колекції.

Поведінка агентів аналізується через журнали рішень, action distribution, state counts, zone visits і route trail. Якщо агент вибирає роботу, у журналі має бути пояснення через розклад, робоче часове вікно або utility-оцінку. Якщо агент переходить до харчування, причина має бути пов’язана зі зниженням hunger. Якщо агент іде в парк або соціальну зону, у журналі має бути відображено вплив happiness, stress або sociality. Така перевірка важлива, тому що правдоподібність симуляції залежить не лише від маршруту, а й від пояснюваності поведінки.

У дослідженнях агентного моделювання важливо перевіряти не лише один ручний сценарій, а і сукупну поведінку багатьох агентів, оскільки агреговані закономірності виникають із локальних рішень окремих учасників [22].

4.3 Перевірка продуктивності та обсягу запису симуляції

Система зберігає повний timeline, тому обсяг даних прямо залежить від тривалості симуляції та кількості агентів. Кількість snapshot визначається формулою:

$$K = T \cdot n \tag{4.1}$$

де K – кількість записів стану агента;

T – кількість tick;

n – кількість агентів.

Для 720 tick і 100 агентів система створює 72000 snapshot, а для 1440 tick і 150 агентів – 216000 snapshot.

Такий обсяг є прийнятним для MVP і дозволяє зберігати детальну історію без додаткової компресії. Для більших симуляцій передбачено напрями оптимізації: зберігання лише подій зміни стану, стиснення timeline, посторінкове завантаження кадрів, попереднє агрегування метрик на backend або використання окремого сховища для часових рядів. У межах роботи повний timeline залишено свідомо, оскільки він спрощує відлагодження, пояснення рішень і повторне відтворення експерименту.

Продуктивність клієнтської частини оцінюється за стабільністю Canvas-рендерингу, швидкістю перемотування timeline, коректністю zoom/pan і відсутністю зависання offcanvas-журналів. Для браузерної перевірки доцільно використовувати Chrome DevTools Performance або аналогічний інструмент, а для загального спостереження за ресурсами – диспетчер задач операційної системи.

4.4 Аналітичні метрики результатів симуляції

Для аналізу результатів сформовано відтворюваний набір CSV-даних для діаграм. Контрольна симуляція має параметри: 100 агентів, seed 42, старт о 07:00, тривалість 720 tick. Дані експортуються у вигляді окремих файлів для зайнятості зон, потреб за профілями, розподілу активностей, переходів між зонами та підсумків маршрутів. Такі графіки адаптують підхід до візуального аналізу міської агентної поведінки, у якому важливо оцінювати не тільки індивідуальні рішення, а й агреговані наслідки багатьох локальних дій [23].

Першою метрикою є теплова карта зайнятості зон. Для кожної години h і типу зони z обчислюється частка snapshot:

$$share(z,h) = \frac{count(z,h)}{\sum_{z \in Z} count(z,h)} \tag{4.2}$$

де $count(z,h)$ – кількість snapshot у зоні z за годину h;

Z – множина типів зон. Ця метрика показує, як агенти протягом дня переміщуються між житловими, робочими, комерційними, освітніми, рекреаційними та медичними зонами.

Теплову карту зайнятості зон наведено на рисунку 4.2.

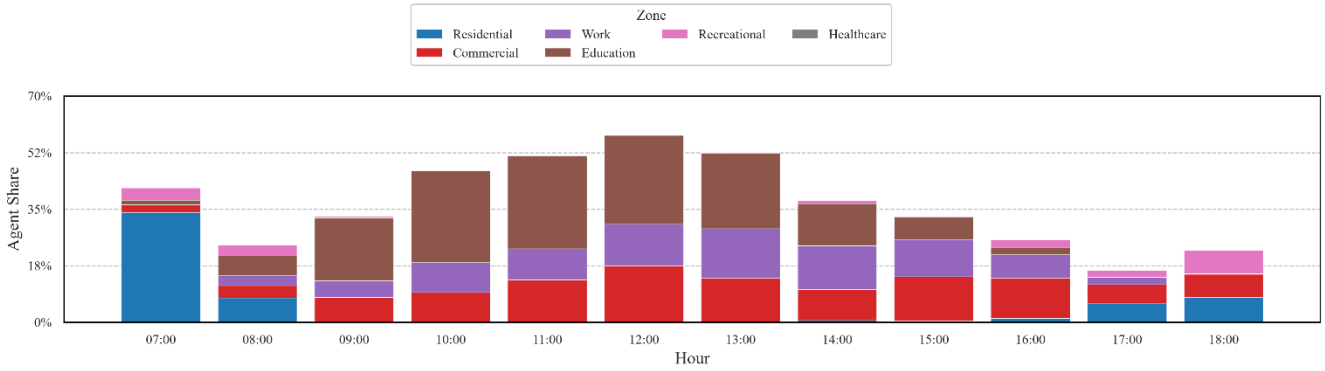


Рисунок 4.2 – Теплова карта зайнятості міських зон за годинами

Другою метрикою є динаміка потреб агентів за поведінковими профілями. Вона дає змогу перевірити, чи потреби змінюються реалістично: energy не повинна падати до нуля масово, hunger має знижуватися поступово, stress повинен зростати під час інтенсивної діяльності й зменшуватися під час відпочинку, productivity має залежати від роботи та стану агента.

Графік потреб наведено на рисунку 4.3.

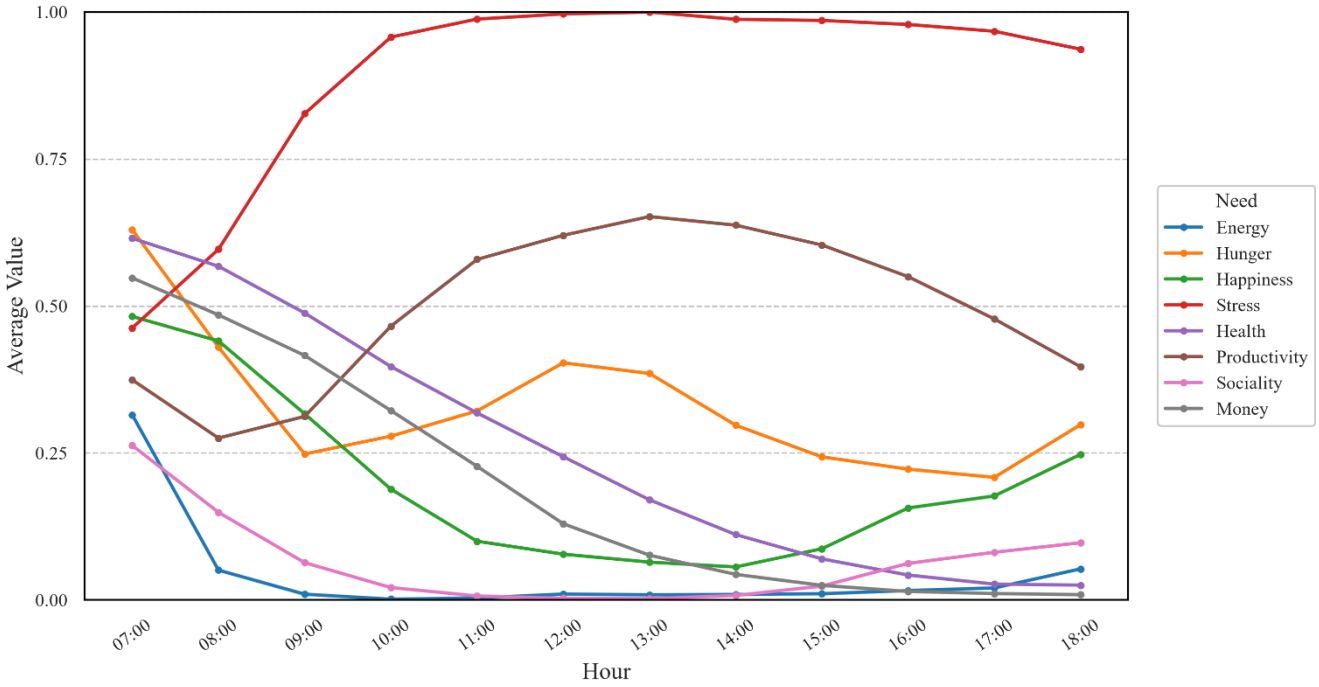


Рисунок 4.3 – Динаміка потреб агентів за поведінковими профілями

Третьою метрикою є розподіл активностей за профілями. Вона показує, чи відрізняються employed, student, health-focused і flexible агенти. Правдоподібна симуляція має демонструвати більшу частку work у employed, більше навчальних активностей у student, більше health або recreation для відповідних профілів і відсутність повної одноманітності між усіма агентами.

Розподіл активностей наведено на рисунку 4.4.

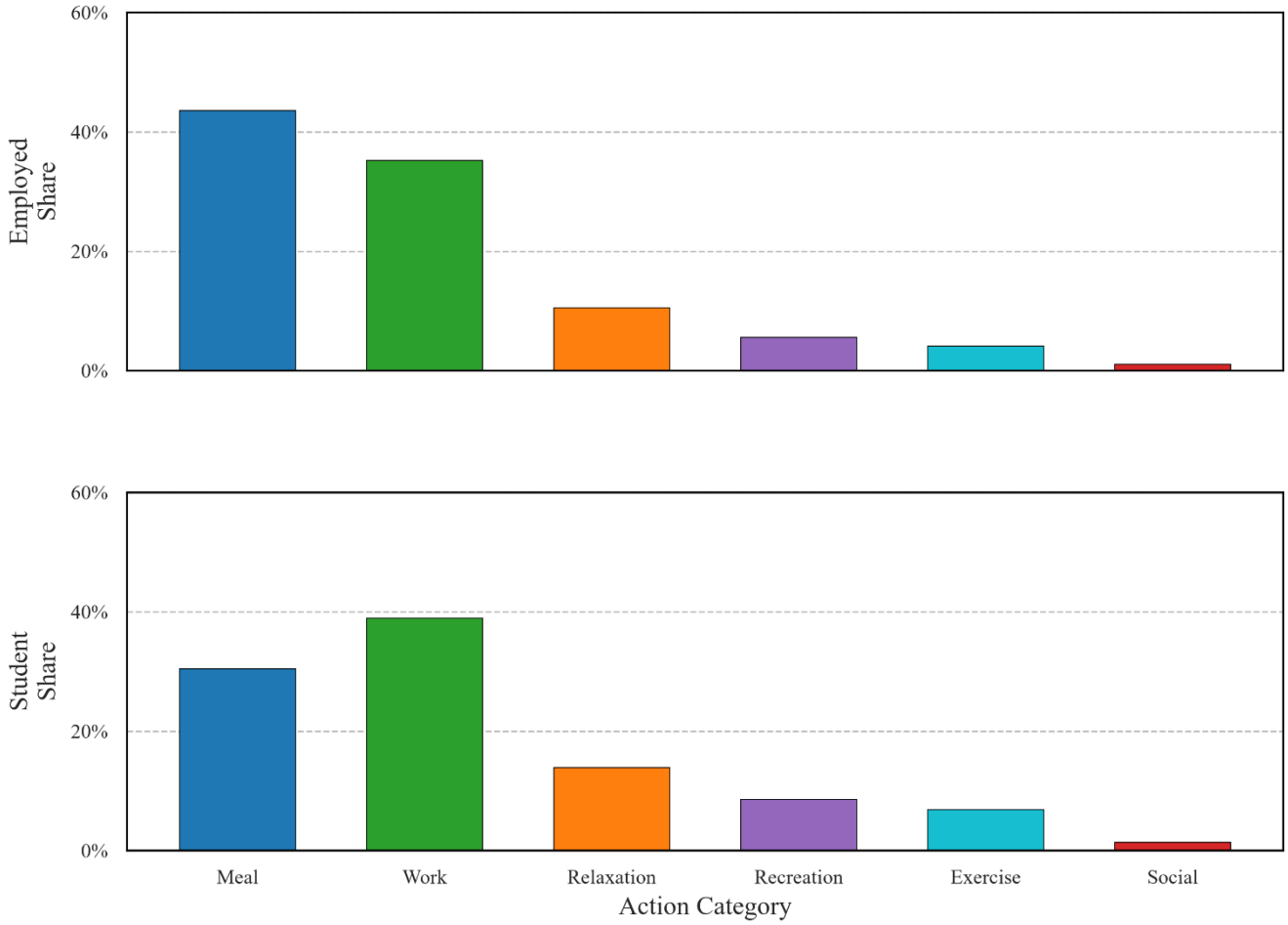


Рисунок 4.4 – Розподіл категорій активностей за профілями агентів.

Четвертою метрикою є матриця переходів між типами зон. Вона показує, які переміщення виникають найчастіше та чи не з’являються неможливі переходи через непрохідні області. Для коректної карти очікувано, що основні переходи проходять через sidewalk і crosswalk, а функціональні зони з’єднуються з мережею руху через entrance tiles.

Матрицю переходів наведено на рисунку 4.5.

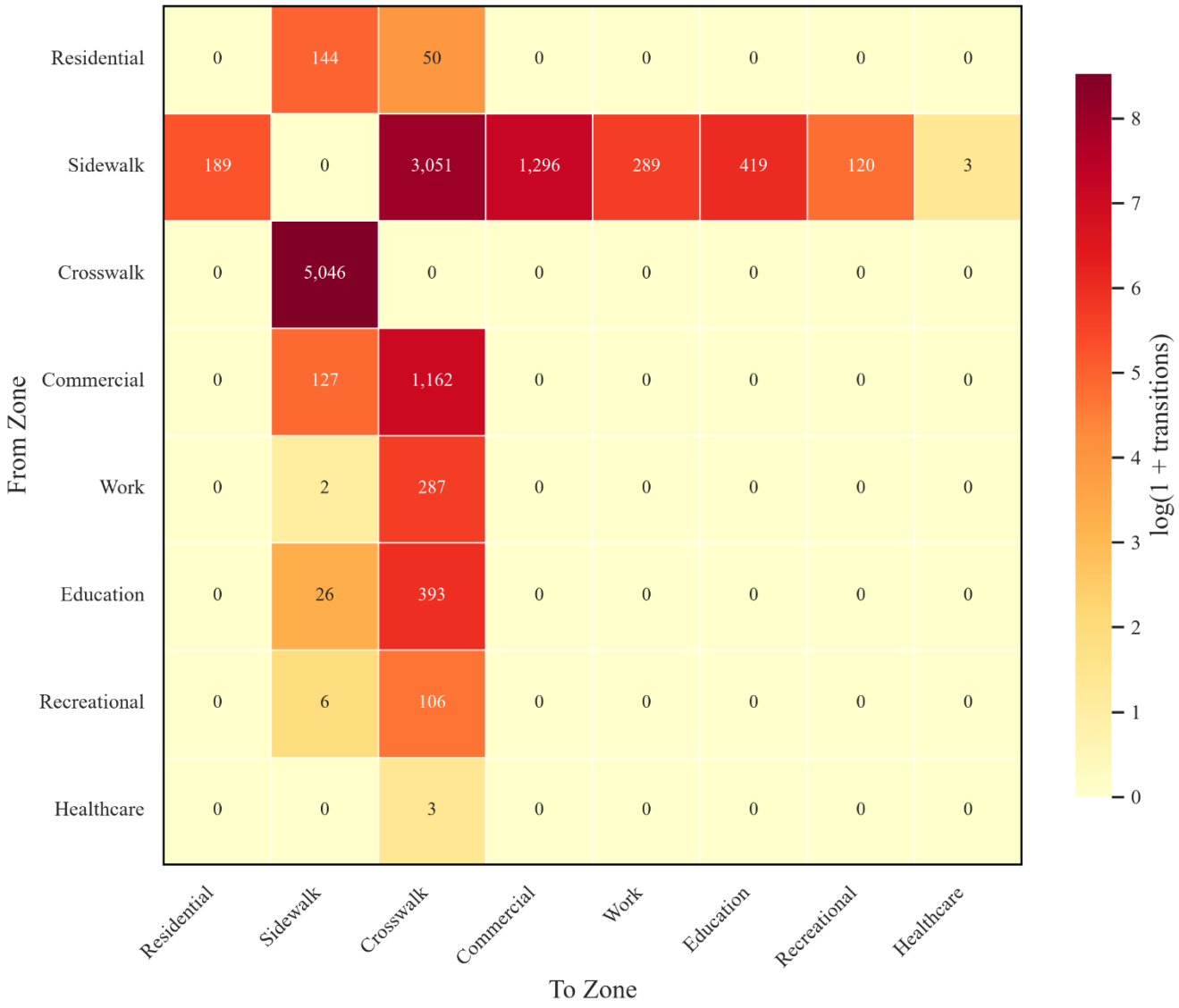


Рисунок 4.5 – Матриця переходів агентів між типами зон

4.5 Підготовка аналітичних візуалізацій

Підготовка аналітичних візуалізацій організована як відтворюваний процес: спочатку генерується контрольна симуляція з фіксованими параметрами, після цього з SQLite-запису експортуються CSV-файли, а вже з них будуються діаграми для розділу аналізу результатів. Така схема дозволяє повторити експеримент після зміни карти, поведінкових правил або кількості агентів і порівняти отримані метрики.

CSV-файли мають просту табличну структуру, тому їх можна відкрити в Excel, LibreOffice Calc, Google Sheets або обробити в Python за допомогою pandas,

matplotlib чи seaborn. У текст роботи вставляються готові рисунки з підписами, легендами та поясненням, а не програмний код побудови графіків. Код експорту та, за потреби, приклади побудови графіків доцільно залишати в репозиторії або додатках.

4.6 Впровадження, системні вимоги та підтримка

Система може запускатися локально через Python або через Docker Compose. Для локального запуску потрібен Python, браузер і доступ до командного рядка. Послідовність запуску: python -m urban_sim init-db, python -m urban_sim generate, python -m urban_sim serve. Для Docker-запуску достатньо виконати docker compose up --build.

Підтримка системи передбачає редагування data/actions.json, data/agents.json, data/sample_map.json і data/scenarios. Якщо змінюється поведінка агентів, основними точками конфігурації є actions і archetype schedules. Якщо змінюється простір міста, редагуються карта, entrance markers, сервіси та властивості тайлів. Якщо система масштабується на більшу кількість агентів, основними напрямами оптимізації є обсяг timeline, API-відповіді, кешування агрегованих метрик і frontend-рендеринг.

5 БЕЗПЕКА ЖИТТЄДІЯЛЬНОСТІ, ОСНОВИ ОХОРОНИ ПРАЦІ

У цьому розділі розглянуто два теоретичні питання з безпеки життєдіяльності та основ охорони праці: моделювання та прогнозування небезпечних ситуацій, а також загальні вимоги безпеки з охорони праці для користувачів ПК. Перше питання належить до сфери безпеки життєдіяльності, оскільки пов’язане з виявленням небезпек, оцінюванням ризику, прогнозуванням розвитку небезпечних подій і запобіганням їхнім негативним наслідкам. Друге питання належить до основ охорони праці, оскільки стосується безпечної організації роботи людини з комп’ютерною технікою, дотримання вимог електробезпеки, пожежної безпеки, режиму праці та відпочинку, а також правильних дій у разі аварійної ситуації.

5.1 Моделювання та прогнозування небезпечних ситуацій

Безпека життєдіяльності є галуззю знань, що вивчає небезпеки, умови їх виникнення, закономірності впливу на людину та середовище, а також способи захисту від негативних наслідків. Її основною метою є забезпечення таких умов існування людини, за яких рівень небезпечних і шкідливих чинників не перевищує допустимих значень, а ризик для життя, здоров’я, працездатності та довкілля залишається прийнятним [24]. Одним із важливих напрямів безпеки життєдіяльності є не лише реагування на небезпеки, а й попереднє прогнозування можливих небезпечних ситуацій.

Небезпека – це явище, процес, об’єкт або умова, що за певних обставин можуть завдати шкоди людині, суспільству, матеріальним цінностям або природному середовищу. Небезпечна ситуація виникає тоді, коли небезпека набуває реального характеру і створює загрозу негативних наслідків. Вона може бути спричинена природними процесами, техногенними аваріями, соціальними факторами, людськими помилками або поєднанням кількох причин. Саме тому для забезпечення безпеки важливо вміти аналізувати можливі небезпеки, встановлювати причини їх виникнення та передбачати розвиток подій [25].

Моделювання небезпечних ситуацій – це створення спрощеного формалізованого опису реальної або можливої небезпечної події з метою її дослідження. Модель дає змогу виділити головні чинники, визначити зв’язки між ними, проаналізувати умови виникнення небезпеки та можливі наслідки. Вона не є повною копією реальності, але відображає ті властивості процесу або об’єкта, які є важливими для конкретного аналізу. У безпеці життєдіяльності моделювання застосовують для вивчення аварій, надзвичайних ситуацій, стихійних лих, техногенних процесів, соціальних небезпек і поведінки людей у складних умовах. [24].

Моделі небезпечних ситуацій можуть бути словесними, графічними, математичними, фізичними, імітаційними та комп’ютерними. Словесна модель подає ситуацію у вигляді опису, інструкції або правил. Графічна модель використовує схеми, карти, діаграми, плани евакуації чи структурно-логічні зв’язки. Математична модель ґрунтується на формулах, рівняннях, імовірнісних залежностях і числових показниках. Фізична модель відтворює процес на спеціальному об’єкті або макеті. Імітаційні та комп’ютерні моделі дозволяють багаторазово відтворювати розвиток подій у часі, змінювати вхідні параметри та порівнювати наслідки різних сценаріїв [25].

Побудова моделі небезпечної ситуації передбачає кілька послідовних етапів. Спочатку визначають мету аналізу та об’єкт дослідження. Далі здійснюють ідентифікацію небезпек, тобто встановлюють можливі джерела загрози, умови їх активізації та об’єкти, на які вони можуть впливати. Після цього обирають показники моделі: час, відстань, швидкість поширення небезпеки, кількість людей, рівень забруднення, температуру, тиск, імовірність відмови обладнання, можливі матеріальні збитки або інші параметри. На наступному етапі визначають зв’язки між елементами моделі, проводять розрахунки або імітацію та аналізують отримані результати.

Одним із ключових понять у моделюванні небезпечних ситуацій є ризик. Ризик характеризує можливість виникнення небезпечної події та тяжкість її наслідків. У загальному розумінні він поєднує ймовірність події та розмір можливої

шкоди. Якщо подія має низьку ймовірність, але може спричинити значні втрати, її не можна ігнорувати. Так само події з незначними наслідками, але високою повторюваністю також потребують уваги, оскільки вони можуть призвести до значних сумарних збитків [25].

Ризик-орієнтований підхід дозволяє визначити, які небезпеки потребують першочергового контролю. Його сутність полягає в тому, що заходи безпеки мають бути спрямовані передусім на ті чинники, які мають найбільшу ймовірність виникнення або найтяжчі наслідки. Для цього виконують виявлення небезпек, оцінювання ймовірності, аналіз можливих втрат, визначення допустимого рівня ризику та вибір заходів для його зниження. Такий підхід застосовується у сфері цивільного захисту, промислової безпеки, охорони праці, екологічної безпеки та управління надзвичайними ситуаціями [24].

Прогнозування небезпечних ситуацій – це науково обґрунтоване передбачення можливого виникнення, розвитку та наслідків небезпечних подій. Воно базується на аналізі статистичних даних, закономірностей розвитку процесів, результатів спостережень, експертних оцінок і даних моделювання. Прогнозування може бути короткостроковим, середньостроковим і довгостроковим. Короткострокове прогнозування застосовують для оперативного реагування, середньострокове – для планування заходів безпеки на певний період, довгострокове – для стратегічного планування розвитку територій, інфраструктури та систем захисту [24].

За способом отримання результатів прогнози можуть бути статистичними, експертними, аналітичними та імітаційними. Статистичне прогнозування ґрунтується на аналізі даних про подібні події в минулому. Експертне прогнозування застосовується тоді, коли точних даних недостатньо, але є можливість використати досвід фахівців. Аналітичне прогнозування базується на математичних залежностях і розрахунках. Імітаційне прогнозування передбачає відтворення можливого розвитку подій за різних початкових умов. На практиці часто поєднують кілька методів, оскільки це підвищує обґрунтованість прогнозу [25].

Особливе значення має прогнозування надзвичайних ситуацій. Надзвичайна ситуація – це обстановка на окремій території чи об’єкті, яка характеризується порушенням нормальних умов життєдіяльності населення та може призвести до людських жертв, матеріальних втрат або шкоди довкіллю. Правові та організаційні засади цивільного захисту населення і територій від надзвичайних ситуацій визначаються Кодексом цивільного захисту України [26]. Відповідно прогнозування таких ситуацій є важливою умовою своєчасного попередження населення, підготовки сил і засобів реагування, організації евакуації та зменшення наслідків небезпечних подій.

Надзвичайні ситуації класифікують за походженням, масштабом і характером наслідків. За походженням вони можуть бути природними, техногенними, соціальними та воєнними. Природні небезпеки пов’язані з повенями, бурями, землетрусами, зсувами, пожежами в екосистемах, засухами або різкими змінами погодних умов. Техногенні небезпеки виникають унаслідок аварій на підприємствах, транспорті, об’єктах енергетики, хімічно небезпечних, радіаційно небезпечних або вибухопожежонебезпечних об’єктах. Соціальні небезпеки можуть бути пов’язані з панікою, масовими заворушеннями, злочинністю або іншими факторами суспільного характеру [24].

Для прогнозування природних небезпек використовують метеорологічні, гідрологічні, сейсмічні та екологічні спостереження, карти небезпечних зон і статистику повторюваності явищ. Для прогнозування техногенних небезпек враховують технічний стан обладнання, рівень зношення основних фондів, наявність небезпечних речовин, дотримання технологічних режимів, людський чинник і стан систем контролю. У багатьох випадках важливим є також урахування поведінки людей, оскільки в умовах небезпеки можливі паніка, неправильне сприйняття інформації або порушення інструкцій.

Результати моделювання та прогнозування використовують для розроблення профілактичних заходів. До них належать усунення або зменшення джерел небезпеки, удосконалення систем контролю, створення резервних систем, навчання персоналу й населення, підготовка планів евакуації, забезпечення засобів

індивідуального та колективного захисту, організація систем оповіщення, проведення тренувань і підвищення стійкості об’єктів. Головна мета цих заходів – не допустити виникнення небезпечної ситуації або зменшити її наслідки до прийнятного рівня [25].

Отже, моделювання та прогнозування небезпечних ситуацій є важливими складовими безпеки життєдіяльності. Моделювання дозволяє формалізувати небезпеку, визначити основні чинники її виникнення та дослідити можливі наслідки. Прогнозування дає змогу передбачити розвиток подій, своєчасно підготувати заходи захисту та знизити рівень ризику. Ефективне використання цих методів сприяє зменшенню людських, матеріальних і екологічних втрат.

5.2 Загальні вимоги безпеки з охорони праці для користувачів ПК

Охорона праці – це система правових, соціально-економічних, організаційно-технічних, санітарно-гігієнічних і профілактичних заходів, спрямованих на збереження життя, здоров’я та працездатності людини у процесі трудової діяльності [27]. Її основною метою є створення безпечних і нешкідливих умов праці, запобігання виробничому травматизму, професійним захворюванням, аваріям і нещасним випадкам. Вимоги охорони праці поширюються на всі види робіт, зокрема на діяльність, пов’язану з використанням персональних комп’ютерів.

Користувач ПК працює з електронно-обчислювальною технікою, екранними пристроями, клавіатурою, маніпулятором, периферійним і мережевим обладнанням. Хоча така робота не належить до фізично важких, вона може супроводжуватися небезпечними та шкідливими чинниками. До них належать електрична небезпека, пожежна небезпека, зорове напруження, тривале статичне навантаження, інформаційне перевантаження, монотонність роботи, шум від обладнання, несприятливий мікроклімат і порушення режиму праці та відпочинку [28].

Загальні вимоги безпеки передбачають, що до роботи з ПК допускаються особи, які пройшли інструктаж з охорони праці, ознайомлені з правилами безпечної експлуатації обладнання та знають порядок дій у разі аварійної ситуації. Користувач повинен виконувати правила внутрішнього розпорядку, використовувати обладнання лише за призначенням, не виконувати самовільний ремонт електрообладнання та повідомляти відповідальних осіб про несправності. Дотримання цих вимог є необхідною умовою профілактики травматизму й аварій [27].

Перед початком роботи користувач повинен перевірити зовнішній стан обладнання. Необхідно переконатися у справності системного блока або ноутбука, монітора, клавіатури, миші, кабелів живлення, розеток, подовжувачів і мережевих фільтрів. Не допускається використання пристроїв із пошкодженою ізоляцією, відкритими струмопровідними частинами, нестійким корпусом, запахом гару, іскрінням, стороннім шумом або ознаками перегрівання. Якщо виявлено несправність, роботу слід припинити до її усунення відповідальним фахівцем.

Електробезпека є одним із головних напрямів охорони праці користувачів ПК. Комп’ютерна техніка працює від електричної мережі, тому неправильне підключення або експлуатація можуть призвести до ураження електричним струмом, короткого замикання чи пожежі. Користувачу забороняється торкатися електричних роз’ємів вологими руками, використовувати пошкоджені кабелі, самостійно розбирати блок живлення, монітор або інше електронне обладнання, перевантажувати розетки, підключати надмірну кількість пристроїв до одного подовжувача та прокладати кабелі в місцях, де вони можуть бути пошкоджені [28].

Пожежна безпека під час роботи з ПК забезпечується справністю електромережі, недопущенням перевантаження розеток, очищенням обладнання від пилу, контролем температурного режиму пристроїв і дотриманням правил користування електрообладнанням. Пил, заблоковані вентиляційні отвори, несправні вентилятори або тривала робота пристроїв у режимі перегрівання можуть створювати передумови для займання. Після завершення роботи необхідно

коректно вимкнути комп’ютер, монітор та інше обладнання, якщо інше не передбачено режимом експлуатації.

У приміщенні, де використовуються персональні комп’ютери, потрібно підтримувати порядок і забезпечувати вільний доступ до виходів, електричних щитків, засобів пожежогасіння та шляхів евакуації. Забороняється захаращувати проходи, розміщувати легкозаймисті матеріали поблизу електрообладнання, залишати ввімкнені несправні пристрої без нагляду або користуватися обладнанням при появі запаху диму. У разі пожежі працівник повинен повідомити про небезпеку, за можливості знеструмити обладнання, скористатися первинними засобами пожежогасіння без ризику для життя та залишити приміщення згідно з планом евакуації [27].

Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями передбачають організацію роботи з урахуванням впливу дисплея, тривалості роботи, перерв, умов приміщення та стану обладнання [29]. Тривала безперервна робота з екраном може спричинити втому очей, головний біль, зниження уваги та загальну перевтому. Тому користувач повинен дотримуватися встановленого режиму праці та відпочинку, робити перерви й не допускати надмірного безперервного навантаження.

До санітарно-гігієнічних вимог належить підтримання належного мікроклімату, чистоти, освітленості та допустимого рівня шуму в приміщенні. Приміщення має регулярно провітрюватися, а обладнання не повинно створювати надмірного шуму або тепловиділення. Недостатнє провітрювання, підвищена температура, сухість повітря або шум можуть погіршувати самопочуття користувача та знижувати працездатність. Підтримання нормальних умов середовища є важливою складовою профілактики професійної втоми [28].

Під час роботи користувачеві забороняється виконувати дії, які можуть створити небезпеку для нього або інших осіб. Не можна самостійно ремонтувати електрообладнання, відкривати корпуси пристроїв під напругою, торкатися внутрішніх елементів комп’ютера без спеціальної підготовки, користуватися несправною технікою, залишати рідини поруч з обладнанням, переміщувати

ввімкнені пристрої без потреби або закривати вентиляційні отвори. Також не можна ігнорувати повідомлення про перегрівання, несправність батареї, помилки живлення або інші ознаки аварійного режиму.

Організаційні заходи охорони праці передбачають проведення інструктажів, навчання безпечним методам роботи, контроль стану обладнання, своєчасне усунення несправностей і дотримання вимог нормативних документів. Інструктажі можуть бути вступними, первинними, повторними, позаплановими та цільовими. Вони потрібні для того, щоб працівник знав потенційні небезпеки свого робочого місця, правила безпечної експлуатації обладнання та порядок дій у разі аварії [27].

У разі аварійної ситуації користувач ПК повинен діяти швидко, але без паніки. Якщо з’явилися запах гару, дим, іскріння, сильний шум вентилятора, перегрівання або нестабільна робота обладнання, потрібно негайно припинити роботу, за можливості безпечно вимкнути пристрій і повідомити відповідальну особу. Якщо сталося ураження людини електричним струмом, насамперед потрібно припинити дію струму без ризику для рятувальника, викликати допомогу та надати домедичну допомогу відповідно до наявних навичок і встановленого порядку.

Отже, загальні вимоги безпеки з охорони праці для користувачів ПК охоплюють комплекс організаційних, технічних, електробезпечних, пожежних і санітарно-гігієнічних заходів. Їх виконання дозволяє запобігти травмам, аваріям, пожежам, перевтомі та погіршенню здоров’я працівників. Безпечна робота з комп’ютерною технікою потребує справного обладнання, належного інструктажу, дотримання правил експлуатації, правильного режиму праці та відпочинку, а також готовності користувача діяти відповідно до інструкції у разі небезпечної ситуації.

ВИСНОВКИ

У результаті виконання кваліфікаційної роботи розроблено програмну систему мультиагентної симуляції міського середовища. Система реалізує підхід попередньої генерації timeline і подальшого програвання запису у браузері.

У роботі проаналізовано предметну область міського агентного моделювання, визначено вимоги до карти, агентів, дій, маршрутів, журналів, метрик і вебінтерфейсу. Окремо виконано проєктування програмної системи: описано варіанти використання, UML-діаграму класів, послідовності взаємодії, діаграми діяльності, діаграми станів, логічну модель даних і правила узгодженості між моделями. Розроблено тайлову карту з прямокутними зонами, тротуарами, дорогами, переходами та входами.

Реалізовано модель агента з потребами, persona, розкладом, домашньою позицією та діями. Вибір дії здійснюється через домінантну категорію, критичні стани, обов’язкові задачі, cosine similarity, needScore, costScore, utility і softmax.

Реалізовано A*-маршрутизацію по прохідних тайлах, внутрішній рух агентів у зонах з дробовими координатами, розведення агентів у межах зони, SQLite-збереження timeline, decision logs, metrics і agent journals. Реалізовано HTTP API, CLI, вебінтерфейс із Canvas-картою, zoom, pan, timeline, generation modal, offcanvas для агента і зони, глобальною панеллю Metrics/Needs History/Legend та export. Проєкт опубліковано в публічному репозиторії GitHub, що забезпечує збереження коду, історії змін і можливість повторного розгортання.

Для аналітики було підготовлено набір візуалізацій та даних: теплову карту зайнятості зон, графіки потреб за профілями, розподіл активностей і матрицю переходів, що дозволяє оцінити агреговану поведінку агентів.

Коректність основних компонентів перевірено автоматизованими тестами. Це підтверджує працездатність реалізованого MVP і створює основу для подальшого розвитку: інтеграції з реальними GIS-даними, покращення соціальної взаємодії агентів, оптимізації великих timeline, додавання редактора action space і статистичного порівняння кількох симуляцій.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Методичні вказівки до виконання кваліфікаційної роботи бакалавра для здобувачів спеціальності 121 Інженерія програмного забезпечення / укладачі Д. М. Михалик, Г. Б. Цуприк, В. М. Бревус. Тернопіль: ТНТУ ім. І. Пулюя, 2024.

2. Ye X., Bougie N., Yamasaki T., Watanabe N. MobileCity: An Efficient Framework for Large-Scale Urban Behavior Simulation. arXiv:2504.16946v4, 2025. URL: https://arxiv.org/html/2504.16946v4 (date of access: 14.06.2026).

3. Tony-Yip. MobileCity source code repository. GitHub. URL: https://github.com/Tony-Yip/MobileCity (date of access: 14.06.2026).

4. Wooldridge M. An Introduction to MultiAgent Systems. 2nd ed. Wiley, 2009. URL: https://uranos.ch/research/references/Wooldridge_2001/TLTK.pdf (date of access: 14.06.2026).

5. Crooks A., Malleson N., Manley E., Heppenstall A. Agent-Based Modelling and Geographical Information Systems: A Practical Primer. SAGE, 2019. URL: https://sk.sagepub.com/book/mono/preview/agent-based-modelling-and-geographical-information-systems.pdf (date of access: 14.06.2026).

6. Park J. S. et al. Generative Agents: Interactive Simulacra of Human Behavior. Proceedings of UIST, 2023. URL: https://arxiv.org/pdf/2304.03442 (date of access: 14.06.2026).

7. Hart P. E., Nilsson N. J., Raphael B. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. IEEE Transactions on Systems Science and Cybernetics, 1968. URL: https://ai.stanford.edu/~nilsson/OnlinePubs-Nils/PublishedPapers/astar.pdf (date of access: 14.06.2026).

8. Python Software Foundation. Python Documentation. URL: https://docs.python.org/3/ (date of access: 14.06.2026).

9. Python Software Foundation. dataclasses – Data Classes. URL: https://docs.python.org/3/library/dataclasses.html (date of access: 14.06.2026).

10. Tiled Map Editor Documentation. URL: https://doc.mapeditor.org/ (date of access: 14.06.2026).

11. SQLite Documentation. URL: https://www.sqlite.org/docs.html (date of access: 14.06.2026).

12. OWASP Foundation. OWASP Web Security Testing Guide. URL: https://owasp.org/www-project-web-security-testing-guide/ (date of access: 14.06.2026).

13. MDN Web Docs. Canvas API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Canvas_API (date of access: 14.06.2026).

14. MDN Web Docs. Fetch API. URL: https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API (date of access: 14.06.2026).

15. MDN Web Docs. HTML dialog element. URL: https://developer.mozilla.org/en-US/docs/Web/HTML/Element/dialog (date of access: 14.06.2026).

16. MDN Web Docs. Window: requestAnimationFrame method. URL: https://developer.mozilla.org/en-US/docs/Web/API/window/requestAnimationFrame (date of access: 14.06.2026).

17. WHATWG. HTML Living Standard. URL: https://html.spec.whatwg.org/print.pdf (date of access: 14.06.2026).

18. Ecma International. ECMA-404 The JSON Data Interchange Syntax. URL: https://www.ecma-international.org/wp-content/uploads/ECMA-404_2nd_edition_december_2017.pdf (date of access: 14.06.2026).

19. Docker Documentation. URL: https://docs.docker.com/ (date of access: 14.06.2026).

20. Docker Compose Documentation. URL: https://docs.docker.com/compose/ (date of access: 14.06.2026).

21. IEEE Computer Society. Guide to the Software Engineering Body of Knowledge. SWEBOK Guide, Version 4.0a. IEEE Computer Society, 2025. URL: https://ieeecs-media.computer.org/media/education/swebok/swebok-v4.pdf (date of access: 14.06.2026).

22. Wilensky U., Rand W. An Introduction to Agent-Based Modeling. MIT Press, 2015. URL: https://mitpress.mit.edu/9780262731898/an-introduction-to-agent-based-modeling/ (date of access: 14.06.2026).

23. Russell S., Norvig P. Artificial Intelligence: A Modern Approach. 4th ed. Pearson, 2020. URL: https://aima.cs.berkeley.edu/ (date of access: 14.06.2026).

24. Атаманчук П.С. Безпека життєдіяльності : навч. посіб. Київ : Центр учбової літератури, 2020. 276 с.

25. Безпека життєдіяльності та охорона праці : підруч. / В.В. Сокуренко, О.М. Бандурка та ін. Харків : ХНУВС, 2021. 308 с.

26. Кодекс цивільного захисту України від 01.07.2013 року.

27. Андрейчук Н.І. Охорона праці : навч. посіб. / Н.І. Андрейчук, Ю.В. Кіт, С.В. Шибанов, О.В. Шерстньова. Львів : Видавництво Львівська політехніка, 2021. 276 с.

28. Жидецький В.Ц. Охорона праці користувачів комп’ютерів : підручник. Львів : Афіша, 2020. 176 с.

29. НПАОП 0.00-7.15-18. Вимоги щодо безпеки та захисту здоров’я працівників під час роботи з екранними пристроями, від 14.02.2018 року №207.

ДОДАТКИ

ДОДАТОК А

Тези конференції

Міністерство освіти і науки України
Тернопільський національний технічний університет
імені Івана Пулюя
Маріборський університет (Словенія)
Технічний університет в Кошице (Словаччина)
Каунаський технологічний університет (Литва)
Львівський національний університет
імені Івана Франка
Гірничо-металургійна академія ім. Станіслава Сташиця (Польща)
Луцький національний технічний університет
Чернівецький національний університет
імені Юрія Федьковича
Вроцлавський економічний університет (Польща)
Університет технологій та економіки
імені Хелени Ходковської (Польща)
Донбаська державна машинобудівна академія

Студентське наукове товариство

IX МІЖНАРОДНА

студентська науково - технічна конференція

"ПРИРОДНИЧІ ТА ГУМАНІТАРНІ НАУКИ. АКТУАЛЬНІ ПИТАННЯ"

24-25 квітня 2026 р.

(збірник тез конференції)

Тернопіль 2026

УДК 621.326
Штокало А. – ст. гр. СП-42
Тернопільський національний технічний університет імені Івана Пулюя

РОЗРОБКА СИМУЛЯЦІЙНОЇ МОДЕЛІ МІСЬКОГО СЕРЕДОВИЩА

Науковий керівник: канд. фіз.-мат. наук Цебрій О.Р.

Shtokalo A.
Ternopil Ivan Puluj National Technical Univercity

DEVELOPMENT OF A SIMULATION MODEL OF THE URBAN ENVIRONMENT

Supervisor: Candidate of Physical and Mathematical Sciences, O.R. Tsebriy

Ключові слова: симуляція, міське середовище, агентне моделювання, генеративні агенти, штучний інтелект.

Keywords: simulation, urban environment, agent-based modeling, generative agents, artificial intelligence.

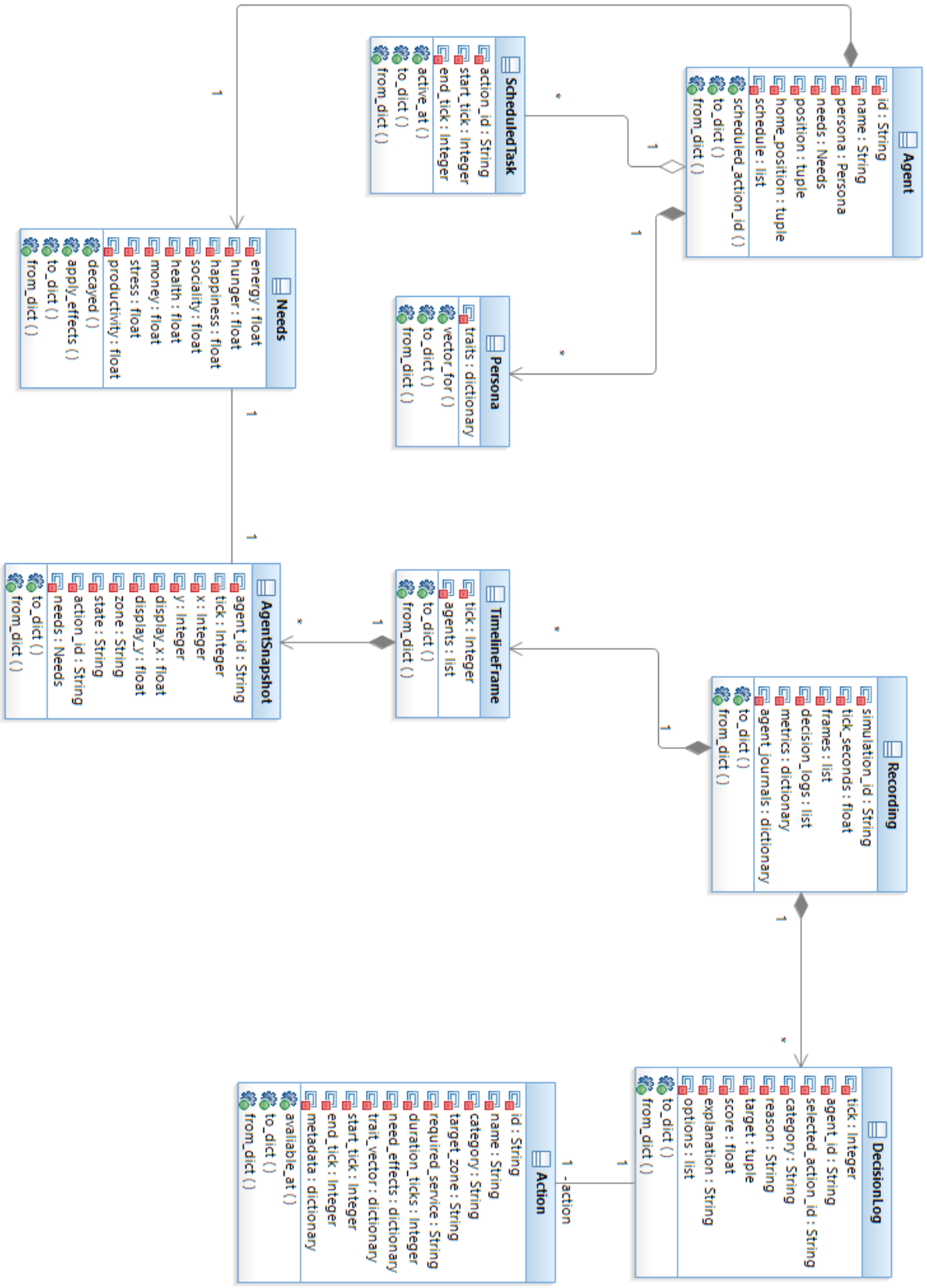
Моделювання поведінки мешканців міста є важливою складовою сучасних досліджень у галузі міського планування та управління транспортними системами. Традиційні агентні моделі зазвичай спираються на жорстко задані правила та обмежені правилами поведінки агенти, що призводить до нереалістичних результатів. Останніми роками великі мовні моделі (LLM) дозволили створювати генеративних агентів – симульованих мешканців із власними потребами, звичками та цілями. Як відзначають Bougie і Watanabe [1], завдяки LLM-агентам можна отримати гнучкішу організацію добового розкладу людини, що краще відображає вплив соціальних та ситуативних чинників. У результаті виникла необхідність у нових платформах для великомасштабної симуляції міських процесів.

У рамках цих досліджень запропоновано декілька підходів до побудови міських симуляцій. Так, Ye і співавт. створили платформу, яка використовує плиткову карту міста з різноманітними будівлями та видами транспорту [2]. Для формування портретів агентів було проведено опитування різних груп населення, що дозволило врахувати демографічні переваги у виборі транспорту та режимах руху. Щоб забезпечити ефективність, автори обмежують вибір дій агентів попередньо згенерованим простором можливих дій і використовують локальні моделі пам’яті для швидкого оновлення внутрішніх станів. Аналогічно, Bougie та Watanabe у фреймворку CitySim запропонували ієрархічне планування щоденних завдань агента, яке балансує обов’язкові справи, особисті звички та зовнішні обставини. Агентам в CitySim також задаються довгострокові цілі, переконання і просторові карти для навігації, що забезпечує складнішу мотиваційну модель і можливість моделювати соціальні взаємодії. Отже, обидва підходи поєднують агентний підхід із ШІ-модулями, дозволяючи відтворювати реалістичні патерни поведінки людей у місті.

Експериментальна оцінка таких симуляцій підтверджує їхню ефективність та реалістичність. У MobileCity було змодельовано понад 4000 агентів, і порівняння з контрольними базовими моделями показало значне покращення якості імітації руху міського населення. Наприклад, MobileCity правильно відображає вибір транспортних засобів та розподіл натовпів по зоні міста, прогнозуючи патерни руху й демографічні

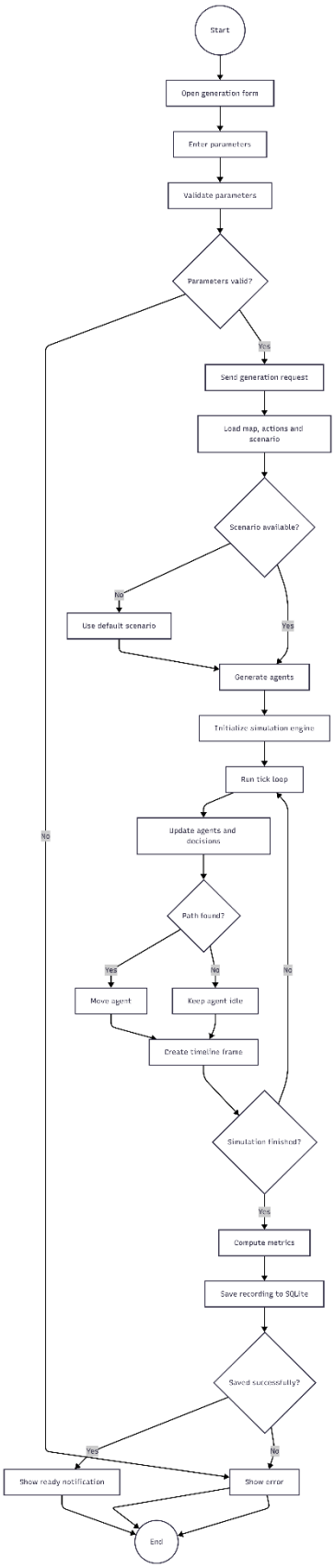
ДОДАТОК Б

Повна UML-діаграма класів



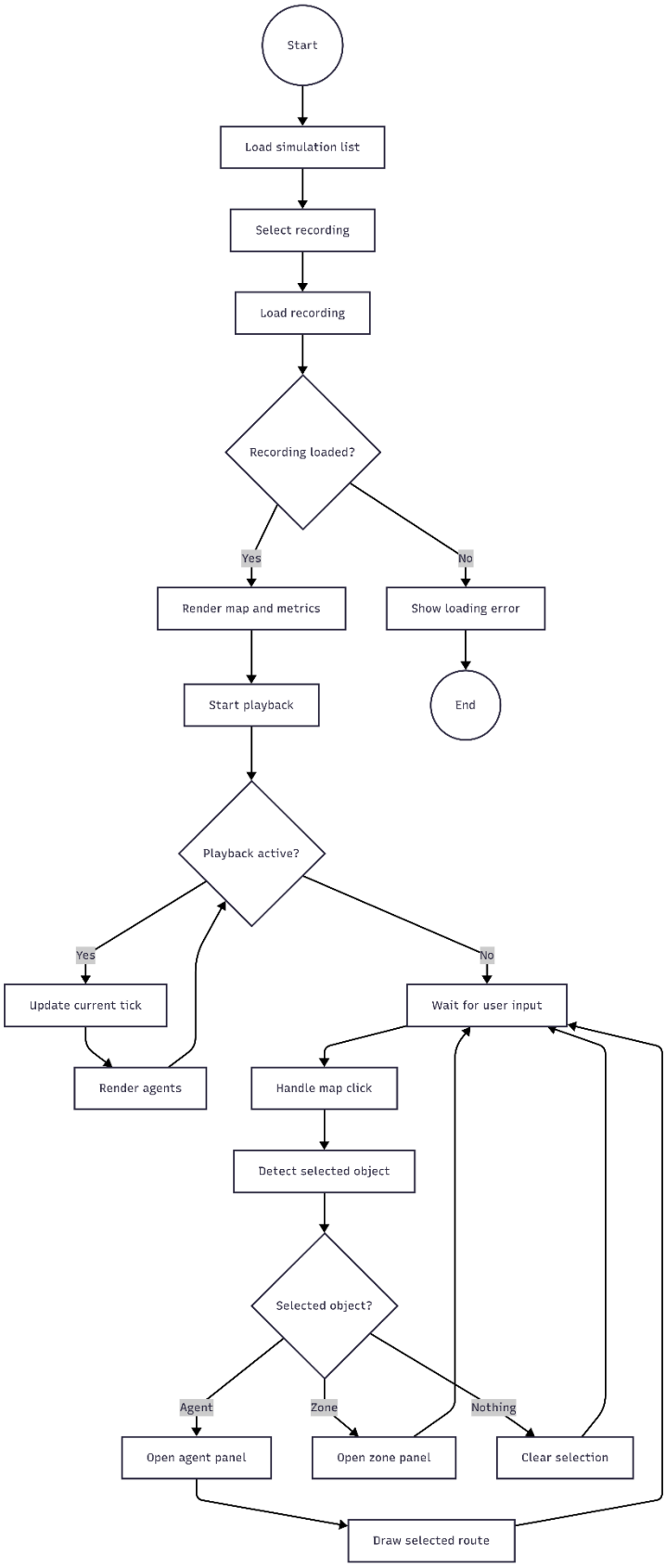
ДОДАТОК В

Діаграма діяльності сценарію генерації симуляції



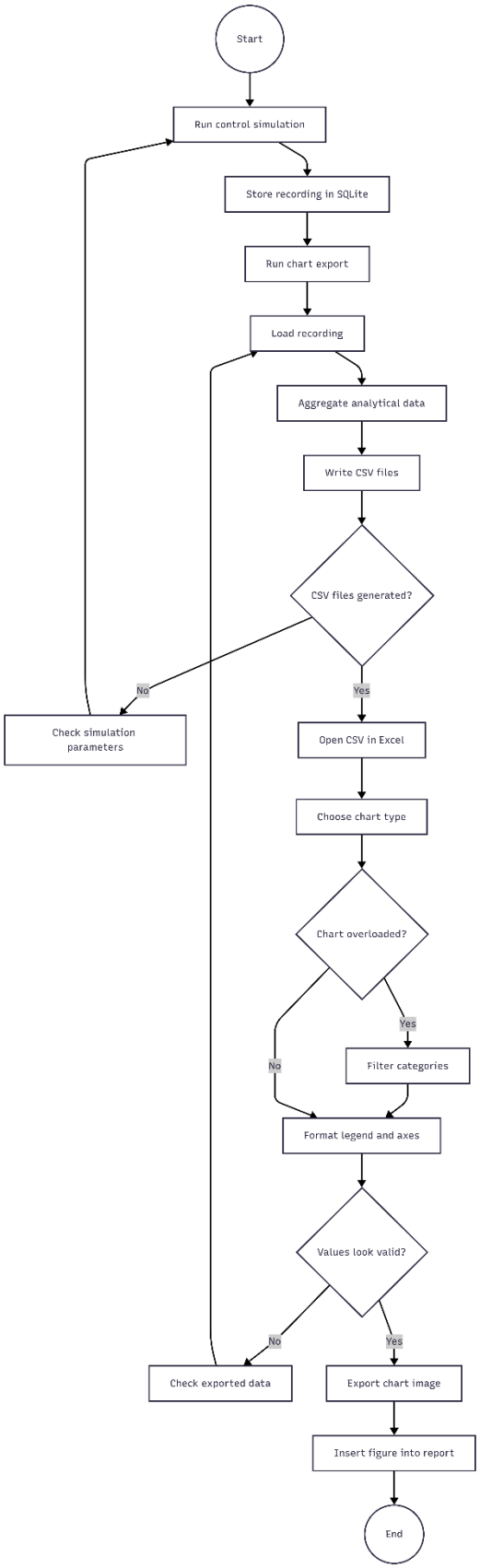
ДОДАТОК Д

Діаграма діяльності сценарію перегляду timeline



ДОДАТОК Е

Діаграма діяльності підготовки аналітичних візуалізацій



ДОДАТОК Ж

Посилання на репозиторій GitHub

https://github.com/but0wka/urban-agent-sim