

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА**

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»**

Кафедра інформаційних систем в економіці

**ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Системи штучного інтелекту»**

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

Форма навчання: очна (денна)

КВАЛІФІКАЦІЙНА МАГІСТЕРСЬКА РОБОТА

на тему: «Розроблення системи автономного управління дорожнього руху
на перехрестях»

здобувача _____ Башкатова Гліба Руслановича
(ПІБ, підпис)

Науковий керівник: _____ к.е.н., доцент, _____ Тішков Б.О.
(науковий ступінь, вчене звання, ПІБ)

(підпис)

**Робота допущена до захисту перед екзаменаційною
комісією з атестації здобувачів вищої освіти (ЕК)**

Завідувач кафедри: _____ к.е.н., доцент, _____ Тішков Б.О.
(науковий ступінь, вчене звання, ПІБ)

(підпис)

Київ 2025

**МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ ЕКОНОМІЧНИЙ УНІВЕРСИТЕТ
ІМЕНІ ВАДИМА ГЕТЬМАНА**

**Навчально-науковий інститут
«Інститут інформаційних технологій в економіці»**

Кафедра інформаційних систем в економіці

**ОСВІТНЬО-ПРОФЕСІЙНА ПРОГРАМА
«Системи штучного інтелекту»**

галузь знань	12 Інформаційні технології
спеціальність	122 Комп'ютерні науки

ПОГОДЖЕНО:
Керівник проєктної групи (гарант)
освітньо-професійної програми

_____ Васильєва Л.В.
« ____ » _____ 2025 р.

ЗАТВЕРДЖУЮ:
Завідувач кафедри інформаційних
систем в економіці

_____ Тішков Б.О.
« ____ » _____ 2025 р.

ІНДИВІДУАЛЬНЕ ЗАВДАННЯ

здобувача вищої освіти _____ Башкатова Гліба Руслановича
(*прізвище, ім'я, по батькові*)

_____ очної (денної) _____ форми навчання
очної (денної), заочної, дистанційної

на підготовку кваліфікаційної магістерської роботи

на тему: **«Розроблення системи автономного управління дорожнього руху
на перехрестях»**

Тему затверджено наказом ректора Університету від «07» березня 2025 р.
№464-ст

Кваліфікаційна магістерська робота виконується на матеріалах

_____ наукових публікацій та матеріалів з мережі інтернет

План кваліфікаційної магістерської роботи

Розділ I Дослідження та аналіз підходів до створення СШІ предметної області
(*назва розділу*)

Розділ II Характеристика СШІ та постановка задачі
(*назва розділу*)

Розділ III Розроблення проєктних рішень.
(назва розділу)

Об'єкт дослідження: система автономного управління дорожнім рухом на перехрестях

Предмет дослідження: методи та алгоритми автономного управління дорожнім рухом із застосуванням інтелектуальних систем

Мета кваліфікаційної магістерської роботи: розробити систему автономного управління дорожнім рухом на перехрестях

Конкретні завдання, які здобувач повинен виконати для досягнення поставленої мети:

У розділі I проводиться аналіз предметної області на основі зібраної інформації, розглядаються існуючі системи штучного інтелекту та інтелектуальні системи у цій сфері, формулюється проблема та визначається задача, обґрунтовується вибір підходів і технологій для проєктування та розробки експертної системи та її компонентів.

У розділі II надається характеристика об'єкта дослідження та формулюється постановка задачі.

У розділі III здійснити проєктування та реалізацію компонентів системи (інформаційного забезпечення, програмного забезпечення, технічного забезпечення). Навести результати реалізації інформаційної системи

Завдання підготував

науковий керівник

(підпис)

Тішков Б.О.

(ініціали, прізвище)

« ____ » _____ 2025 р.

Завдання одержав

здобувач

(підпис)

Башкатов Г.Р.

(ініціали, прізвище)

« ____ » _____ 2025 р.

Реферат

Кваліфікаційна магістерська робота містить 65 сторінок, 39 рисунків, список використаних джерел з 40 найменувань, 2 додатки.

«Розроблення системи автономного управління дорожнього руху на перехрестях»

Об'єктом дослідження кваліфікаційної магістерської роботи є система автономного управління дорожнім рухом на перехрестях.

Предметом дослідження є методи та алгоритми автономного управління дорожнім рухом із застосуванням інтелектуальних систем.

Мета і завдання дослідження. Основною метою кваліфікаційної магістерської роботи є вдосконалення процесу управління дорожнім рухом на перехрестях шляхом розробки інтелектуальної інформаційної системи.

Відповідно до поставленої мети визначені такі завдання:

- дослідити предметну область, виконати аналіз існуючих систем автономного управління дорожнім рухом на перехрестях;
- представити обґрунтування вибору підходів і технологій для створення системи;
- визначити структуру та представити характеристику системи автономного управління дорожнім рухом на перехрестях;
- описати методи та моделі в системі автономного управління дорожнім рухом на перехрестях;
- спроектувати базу знань для системи автономного управління дорожнім рухом на перехрестях;
- описати програмне забезпечення для системи автономного управління дорожнім рухом на перехрестях;
- описати технічне забезпечення для системи автономного управління дорожнім рухом на перехрестях;
- представити результати реалізації системи автономного управління дорожнім рухом на перехрестях.

Теоретична, методична та практична значущість отриманих результатів. Під час дослідження було досліджено предметну область, проаналізовано варіанти зберігання, обробки та управління даними про транспортні потоки на перехрестях, визначено перспективність удосконалення процесу організації дорожнього руху за допомогою розробки спеціалізованого програмного забезпечення – системи автономного управління дорожнім рухом на перехрестях.

Практичні результати дослідження полягають у вдосконаленні процедури управління дорожнім рухом, що призведе до підвищення ефективності роботи транспортної інфраструктури та безпеки на перехрестях.

Рік виконання кваліфікаційної магістерської роботи – 2025.

Рік захисту роботи – 2025.

Ключові слова: машинне навчання, навчання з підкріпленням, інтелектуальна система, автономне управління, дорожній рух, перехрестя, симуляція, візуалізація.

Summary

Qualification master's thesis contains 65 pages, 39 figures, a list of references from 40 titles, 2 applications.

«Development of an autonomous traffic management system at intersections»

The *object of the master's qualification thesis* is the autonomous traffic control system at intersections.

The *subject of the study* is the methods and algorithms of autonomous traffic control using intelligent systems.

Purpose and objectives of the study. The main aim of the master's qualification thesis is to improve the traffic control process at intersections by developing an intelligent information system.

The objectives to achieve this aim are:

- study the subject area and analyse existing autonomous traffic control systems at intersections;
- justify the choice of approaches and technologies for building the system;
- define the structure and describe the characteristics of the autonomous traffic control system at intersections;
- describe the methods and models used in the autonomous traffic control system;
- design a knowledge base for the autonomous traffic control system;
- describe the software for the autonomous traffic control system;
- describe the hardware for the autonomous traffic control system;
- present the implementation results of the autonomous traffic control system.

Theoretical, methodological and practical significance of the results. The study explored the subject area, analysed options for storing, processing, and managing data on traffic flows at intersections, and identified the potential for improving traffic organization through the development of specialized software—an autonomous traffic control system at intersections.

The practical results of the study include improvements in traffic control procedures, leading to increased efficiency of the transport infrastructure and safety at intersections.

The year of completion of the master's thesis is 2025.

The year of defence is 2025.

Keywords: machine learning, reinforcement learning, intelligent system, autonomous control, traffic, intersection, simulation, visualization.

В і д г у к
про кваліфікаційну магістерську роботу
здобувача навчально-наукового інституту «Інститут інформаційних технологій в
економіці»
освітньо-професійної програми «Системи штучного інтелекту»

(прізвище, ім'я, по батькові)

на тему «_____»

1. Актуальність теми: _____

2. Позитивні риси кваліфікаційної магістерської роботи: _____

3. Наявність самостійних розробок автора _____

4. Цінність теоретичних висновків та практичних рекомендацій: _____

5. Наявність недоліків: _____

6. Загальна оцінка кваліфікаційної магістерської роботи та її допущення до захисту перед ЕК: _____

Науковий керівник _____

(посада, учене звання, науковий ступінь)

(підпис)

(прізвище, ініціали)

« ____ » _____ 2025 р.

Рецензія
на кваліфікаційну магістерську роботу
здобувача вищої освіти
Башкатова Гліба Руслановича

Тема «Розроблення системи автономного управління дорожнього руху на перехрестях»

Актуальність теми кваліфікаційної магістерської роботи і доцільність її розроблення. Тема роботи є надзвичайно актуальною, особливо з огляду на потребу в підвищенні ефективності управління дорожнім рухом у сучасних містах. Впровадження інтелектуальних систем, які здатні адаптовуватися до змін транспортного потоку в реальному часі, є важливою складовою розвитку розумної інфраструктури.

Якість проведеного дослідження. Робота відзначається високим рівнем аналітичної глибини, чіткою логікою викладення матеріалу та обґрунтованістю обраних технічних рішень. Автором проведено детальний аналіз предметної області, підібрано відповідні алгоритми підкріплювального навчання, а також здійснено моделювання процесів в середовищі SUMO з метою підтвердження ефективності запропонованого підходу.

Позитивні риси кваліфікаційної магістерської роботи. Робота характеризується якісною структурою, вдалим поєднанням теоретичних та практичних аспектів, використанням сучасних методів штучного інтелекту, грамотним оформленням та наочністю матеріалів. Здобувач продемонстрував вміння аналізувати складні технічні задачі та реалізовувати їх на практиці.

Зауваження. До роботи є незначні зауваження переважно стилістичного характеру. У майбутньому доцільно було б розширити дослідження за рахунок використання реальних даних з міських перехресть та розглянути питання інтеграції з реальним апаратним забезпеченням.

Практична значимість висновків і рекомендацій. Отримані результати мають значний практичний потенціал. Розроблена система може бути застосована в умовах сучасної міської інфраструктури для покращення роботи транспортної мережі.

Місце роботи та посада рецензента

Науковий ступінь, учене звання (за наявності) _____
(підпис, ПІБ)

Підпис засвідчую: _____
(посада, підпис)

ЗМІСТ

ВСТУП.....	3
РОЗДІЛ 1 ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ СШІ ПРЕДМЕТНОЇ ОБЛАСТІ	5
1.1 Дослідження предметної області. Збір інформації та вивчення матеріалів	5
1.2 Аналіз існуючих СШІ й інтелектуальних систем предметної області	9
1.3 Постановка проблеми та формування задач.....	12
1.4 Обґрунтування вибору підходів і технологій для створення СШІ.....	16
Висновки до Розділу 1	21
РОЗДІЛ 2 ХАРАКТЕРИСТИКА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ТА ПОСТАНОВКА ЗАДАЧ.....	22
2.1 Характеристика об'єкта дослідження	22
2.2 Структура і характеристика СШІ.....	27
2.3 Принципи та алгоритми для оптимізації дорожнього руху	29
Висновки до Розділу 2	34
РОЗДІЛ 3 РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ	35
3.1 Загальний підхід до створення системи.....	35
3.2 Визначення симуляційного середовища	38
3.3 Визначення параметрів моделі.....	44
3.4 Тренування та порівняння моделей.....	46
3.5 Візуалізація та перевірка результатів	55
Висновки до Розділу 3	59
ВИСНОВКИ.....	61
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ	63
ДОДАТКИ	66

ВСТУП

Актуальність теми. Зростання автомобільного трафіку в містах України зумовлює необхідність впровадження новітніх технологій в управління дорожнім рухом. Сучасні світлофори та системи регулювання не завжди забезпечують ефективну пропускну здатність, особливо на завантажених перехрестях. Автономні системи управління здатні адаптуватися до змін дорожньої ситуації в реальному часі, що робить їх актуальними для підвищення безпеки та ефективності транспортної інфраструктури. Така тема є особливо важливою для України в умовах урбанізації та розвитку «розумних» міст.

Аналіз останніх досліджень і публікацій. Останні дослідження у сфері автономного управління дорожнім рухом демонструють ефективність методів машинного навчання, зокрема підкріплювального, для адаптивного регулювання світлофорів. Розробки охоплюють виявлення транспортних засобів, адаптацію до змін трафіку та забезпечення стійкості до збоїв. Водночас, ці підходи здебільшого протестовані лише в симуляціях, без урахування реальних умов. Українська компанія SEA пропонує комплексні рішення для управління дорожнім рухом з розвиненою інфраструктурною складовою, проте без повноцінної автономності. Невирішеними залишаються задачі впровадження автономного керування в реальних міських умовах, адаптація до нестабільного трафіку та інтеграція інтелектуальних алгоритмів у ресурсообмежене обладнання.

Мета і завдання дослідження. Метою роботи є розроблення системи автономного управління дорожнім рухом на перехрестях з використанням алгоритмів штучного інтелекту. Для досягнення цієї мети необхідно:

- провести аналіз існуючих систем управління трафіком;
- розробити архітектуру автономної системи;
- реалізувати алгоритм прийняття рішень на основі потоку транспорту;
- створити програмний прототип системи.

Об'єкт дослідження – система автономного управління дорожнім рухом на перехрестях.

Предмет дослідження – методи та алгоритми автономного управління дорожнім рухом із застосуванням інтелектуальних систем.

Методи дослідження. У роботі використано методи системного аналізу для опису структури проблеми, методи імітаційного моделювання – для тестування алгоритмів, а також методи машинного навчання – для адаптації системи до змін трафіку.

Теоретична, методична та практична значимість. Результати роботи мають практичне значення для впровадження в інтелектуальні транспортні системи. Розроблений алгоритм може бути застосований у системах управління рухом міст України. Методична значимість полягає в уніфікованому підході до аналізу та проєктування автономних керувальних систем, що може бути використаний для інших об'єктів транспортної інфраструктури.

Інформаційна база дослідження. До інформаційної бази належать наукові праці з інтелектуальних транспортних систем, автономних систем контролю руху, статистичні дані з офіційних джерел, а також технічна документація щодо апаратних засобів та сенсорів.

Структура роботи. Кваліфікаційна робота складається зі вступу, трьох розділів, висновків, списку використаних джерел та додатків. У першому розділі подано теоретичні засади автоматизованого управління дорожнім рухом. Другий розділ присвячено аналізу та проєктуванню системи. У третьому – реалізовано та протестовано розроблений прототип.

РОЗДІЛ 1

ДОСЛІДЖЕННЯ ТА АНАЛІЗ ПІДХОДІВ ДО СТВОРЕННЯ СШІ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1 Дослідження предметної області. Збір інформації та вивчення матеріалів

Зростання міського транспорту та прискорена урбанізація призводять до значного навантаження на дорожню мережу. Згідно зі статистикою, нерегульовані або фіксовані цикли світлофорів можуть додавати до 10% часу затримок у русі автомобілів. Це зумовлює потребу в адаптивних системах управління світлофорами, які в реальному часі регулюють тривалість фаз залежно від інтенсивності руху. Перші системи регулювання передбачали ручний режим, де інспектори самостійно змінювали часові інтервали сигналів, що дало обмежені можливості реагувати на динамічні зміни потоку. Подальший розвиток привів до автоматизованих систем (наприклад, SCOOT [1], SCATS [2]), що використовують сенсори та центральне керування для адаптивного перерахунку таймінгів.

В Україні впровадження таких рішень лише набирає обертів, проте вже є успішні приклади реалізації. Одним із провідних постачальників світлофорного обладнання та рішень у цій галузі є компанія SEA, досвід якої можна розглядати як репрезентативний для аналізу поточного стану й перспектив розвитку інтелектуальних транспортних систем у країні.

Компанія SEA розробляє сучасні рішення для світлофорної галузі. Їхня Автоматизована система керування дорожнім рухом (АСКДР) SEA TCS забезпечує централізоване й координоване регулювання трафіку [3]. За допомогою SEA TCS можна одночасно регулювати роботу світлофорних об'єктів, контролювати працездатність обладнання та керувати вуличним освітленням. Впровадження комплексу SEA дозволяє зменшити завантаженість перехресть, забезпечити зв'язок

із контролерами та оперативний нагляд за їхнім станом, а також знизити рівень шуму й викидів в околицях перехресть.

Обробка даних у автоматизованій системі керування дорожнім рухом здійснюється за допомогою програмного забезпечення АСКДР SEA TCS, яке побудовано за клієнт-серверною архітектурою (рис. 1.2).

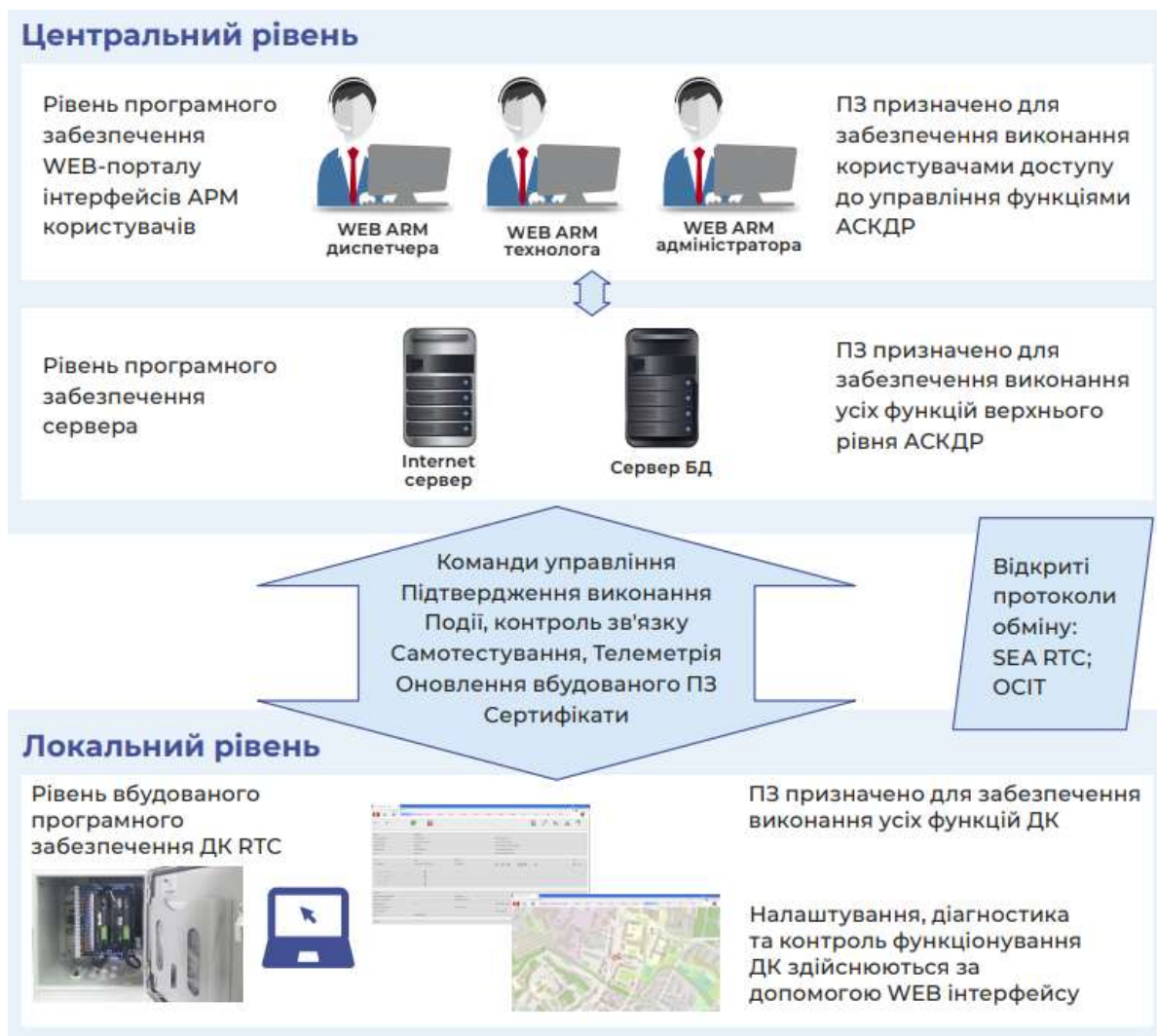


Рисунок 1.1 – Архітектура АСКДР SEA TCS

Джерело: [4]

Основні функції АСКДР включають локальне й централізоване управління світлофорами, диспетчеризацію окремих об'єктів та груп сигналів, постійний моніторинг стану периферійного обладнання (контролерів RTC, світлофорів тощо) та відображення фаз у реальному масштабі часу. Також реалізовано режим «зелена

хвиля», що координує роботу світлофорів, інформаційних табло та пріоритетних маршрутичних ліній. Апаратуру системи складають LED-світлофори та дорожні контролери нового покоління з можливістю дистанційного управління через інтернет. Відповідно, стандартний комплект SEA на перехресті містить камеру та ПК з виходом у мережу, що дозволяє збирати реальні дані про рух і централізовано регулювати світлофори (рис. 1.2).

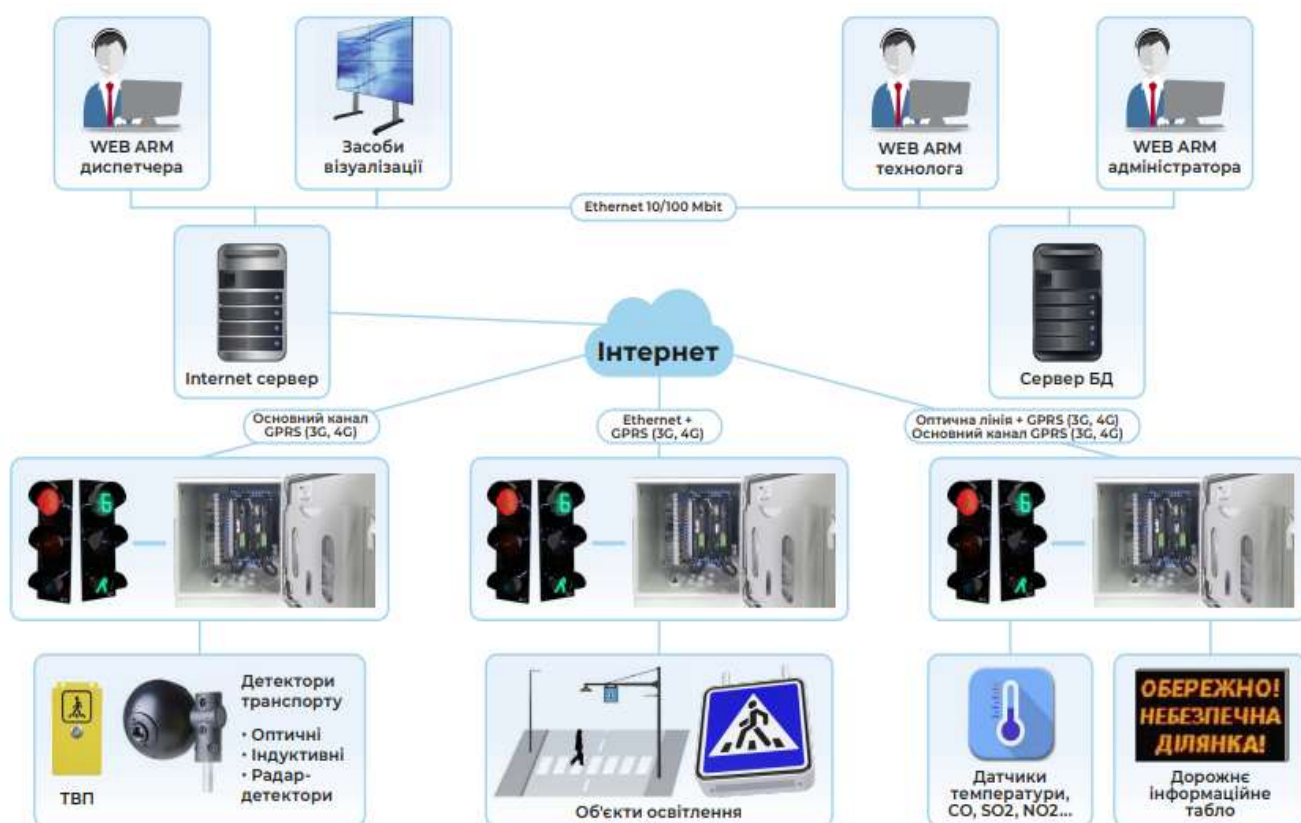


Рисунок 1.2 – Стандартна комплектація системи SEA

Джерело: [3]

Дорожні контролери RTC, що розробляються компанією SEA, є високотехнологічними пристроями з відкритою архітектурою, побудованою на базі промислового комп'ютера. Такий підхід не лише забезпечує високу надійність у складних умовах експлуатації, а й створює широкі можливості для розширення функціоналу контролера в залежності від потреб конкретного об'єкта. Завдяки цьому контролери можуть бути адаптовані під різні сценарії управління дорожнім

рухом, включно з підтримкою нових протоколів зв'язку, підключенням додаткових датчиків і сенсорів, а також розгортанням більш складного програмного забезпечення.

Інтеграція промислового комп'ютера дозволяє реалізувати обробку даних та прийняття рішень безпосередньо на рівні контролера. Це відкриває перспективу розробки та впровадження систем штучного інтелекту, які функціонують локально без постійного звернення до центрального серверу. Наприклад, нейронна мережа, навчена на історичних та поточних даних про інтенсивність руху, може оперативно коригувати сигнальні фази в режимі реального часу. Такий варіант є особливо ефективним у випадках тимчасової втрати з'єднання з централізованою АСКДР – система продовжує працювати автономно, не втрачаючи адаптивності.

Для перехресть з підвищеним навантаженням або складною геометрією можливе застосування комбінованої схеми. У ній складні моделі обробки, що вимагають значних обчислювальних ресурсів, розгортаються на серверній частині АСКДР, тоді як контролери (рис. 1.3) отримують зменшені версії моделей або просто керуючі сигнали. Такий мікс дозволяє зберігати централізований контроль і одночасно забезпечити стійкість системи до мережеских збоїв



Рисунок 1.3 – Дорожний контролер

Джерело: [5]

Таким чином, використання промислових комп'ютерів у складі RTC-контролерів створює технічну базу для переходу до більш автономних, інтелектуальних систем управління дорожнім рухом, які можуть ефективно функціонувати як у централізованому, так і в локальному режимі.

1.2 Аналіз існуючих СШІ й інтелектуальних систем предметної області

Сучасні «розумні» системи дедалі активніше застосовують методи комп'ютерного зору та штучного інтелекту. Зокрема, у роботі [6] запропонували рішення на основі глибокої мережі Faster R-CNN для сегментації транспортних засобів на відеопотоках (рис. 1.2). Підхід складається з етапів побудови адаптивної моделі фону, застосування підмережі Faster R-CNN для виділення об'єктів, а також оптимізації результатів за допомогою топологічних активних мереж. Отримана модель ефективно справляється з складними умовами (затор, затемнення) і забезпечує високу точність визначення положення автомобілів, що є важливим для оцінки завантаженості перехресть.



Рисунок 1.4 – Ідентифікація автомобілів на дорозі

Джерело: [6]

В дослідженні показали, що сегментація транспортних засобів відіграє важливу роль у «розумному» управлінні; вони застосовують Faster R-CNN для глибокої сегментації авто та топологічні активні мережі для уточнення кордонів об'єктів [7]. Підхід успішно справляється з ускладненнями (оклюзіями, змінами освітлення) і перевершує традиційні методи за точністю. Проте ця робота зосереджена на виявленні і покращенні розпізнавання, а не на безпосередньому регулюванні світлофорів.

Щодо алгоритмів управління трафіком, часто використовують симулятори, зокрема, SUMO (Simulation of Urban Mobility) дозволяє моделювати великі міські мережі в реальному часі [8].

У [9] дослідили надійність адаптивної системи управління світлофорами, побудованої на підкріплювальному навчанні. За допомогою симулятора SUMO вони протестували декілька RL-моделей на двох сценаріях [штучна мережа 4×4 та реальна мережа Інгольштадта, Німеччина (рис. 1.5)] у різних умовах (ранковий та вечірній трафік, події, перекриття доріг, відмови сенсорів тощо).

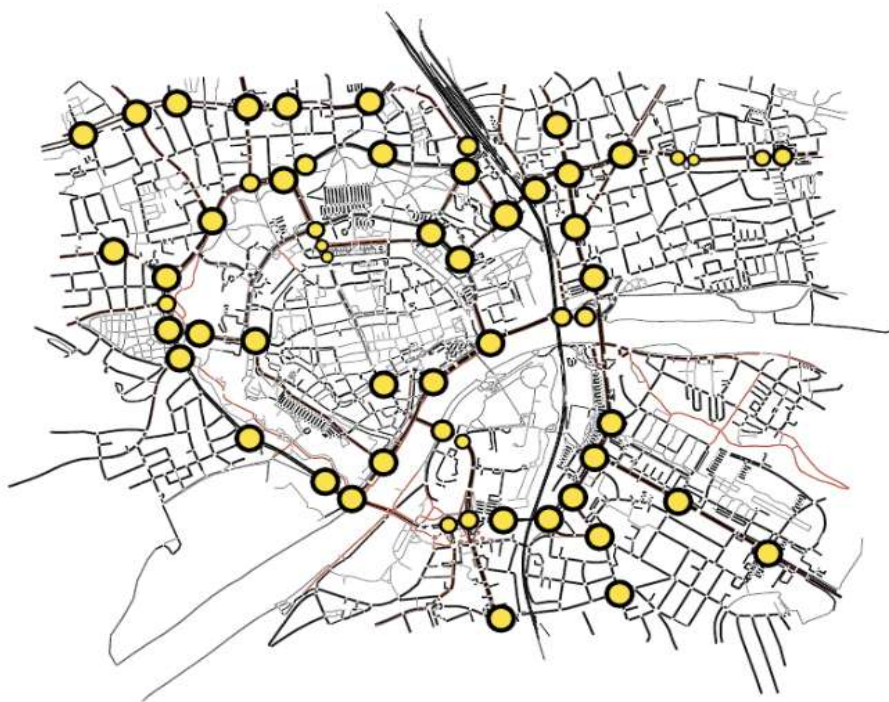


Рисунок 1.5 – Мережа Інгольштадта

Виявилося, що алгоритми RL доволі добре адаптуються до стандартних змін у трафіку, однак зазнають суттєвого погіршення продуктивності при неочікуваних подіях (наприклад, блокування дороги або збій датчиків). Критеріями оцінювання слугували час відновлення ефективної роботи після аномалії та середня довжина черг, і результати показали, що тривале відновлення призводить до значного накопичення заторів. Це свідчить про необхідність впровадження резервних механізмів або додаткової обробки сигналів у реальній системі.

Також у комплексному огляді [10] класифікуються наявні AI-методи для адаптивного управління світлофорами за типом алгоритму (розумні, нечіткі, метаевристичні, RL, DRL тощо). Авторами зроблено висновок, що сучасні адаптивні системи, спроектовані із застосуванням цих підходів, забезпечують суттєве покращення керування трафіком порівняно зі статичними розкладом світлофорів. Особливо відзначено, що гібридні рішення на базі RL/DRL успішно регулюють інтенсивність руху в умовах змінного попиту, хоча масштабування на весь міський масив залишається важливим напрямом подальших досліджень.

Ще одним прикладом є [11] – система для індійських міських умов, що об'єднує детекцію і класифікацію транспортних засобів із динамічним регулюванням світлофорів. Тут застосовано декілька CNN-моделей (VGG16, MobileNetV2, ResNet50) для обробки зображень з камер та визначення типу і кількості транспорту, а власний алгоритм коригує час зеленої фази залежно від щільності і типу потоку (рис. 1.6). У моделюванні реальних сценаріїв їхня система показала значні покращення потоків порівняно зі статичними схемами, при цьому точність детекції авто досягала понад 98%. Отже, інтеграція комп'ютерного зору і адаптивного керування має очевидний потенціал, хоча результати залежать від якості навчаного датасету та реалістичності тестових умов.

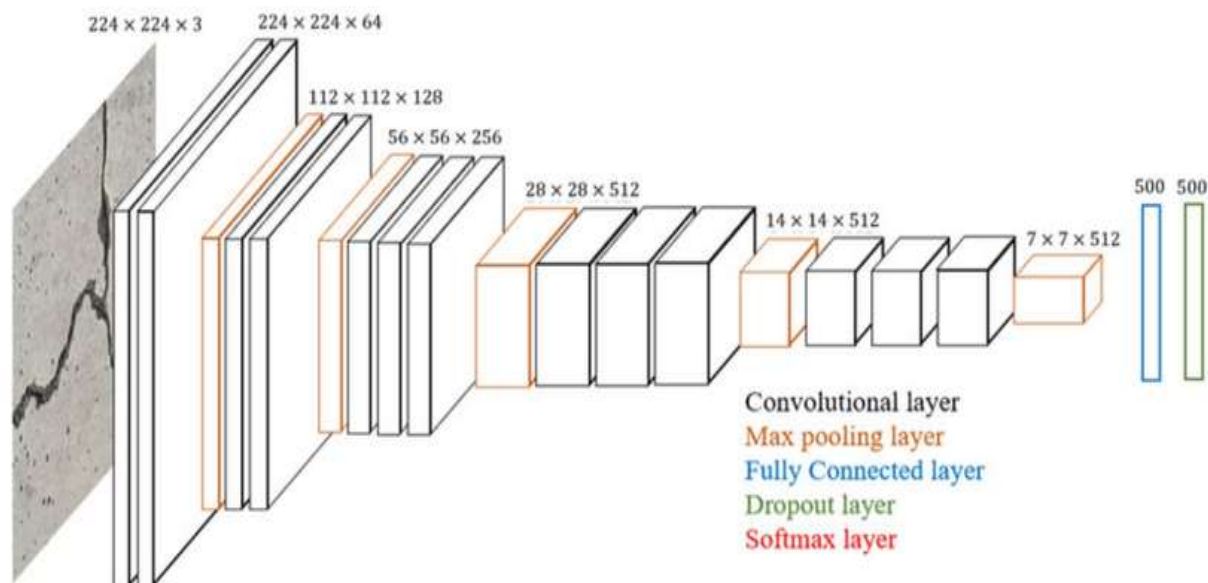


Рисунок 1.6 – Архітектура моделі VGG16

Джерело: [11]

Підсумовуючи, існуючі системи демонструють різноманітні підходи: від класичних АСКДР з централізованим контролем (як SEA TCS) до інноваційних алгоритмів ML/DL. Водночас на сьогодні не існує широко впровадженої автономної системи світлофорного керування, що комбінувала б стандартне обладнання з RL-алгоритмами. Це відкриває перспективу розробки рішення, яке поєднуватиме вже встановлене обладнання SEA (камери, контролери, ПК) з методами підкріплювального навчання.

1.3 Постановка проблеми та формування задач

Сучасна міська інфраструктура управління дорожнім рухом, навіть за наявності передового обладнання, як-от у системах компанії SEA (LED-світлофори, дорожні контролери нового покоління з можливістю мережевого зв'язку), часто не реалізує повний потенціал адаптивного управління. Наявність на перехресті камери, підключеної до ПК з доступом в інтернет, створює технічну можливість для збору даних про транспортний потік у реальному часі. Однак, традиційні системи

з фіксованими циклами або обмежено адаптивними алгоритмами не завжди здатні ефективно реагувати на динамічні зміни трафіку, що призводить до значних негативних наслідків.

Економічні втрати. Затори та надмірні затримки на перехрестях призводять до марної витрати пального транспортними засобами, що збільшує операційні витрати як для приватних автовласників, так і для комерційного транспорту. Втрачається робочий час людей, що змушені більше перебувати в дорозі, знижуючи загальну продуктивність економіки. Також зростають витрати на технічне обслуговування автомобілів через часті зупинки та рушання. Затримки в русі громадського транспорту та служб доставки негативно впливають на ефективність бізнес-процесів та логістику.

Екологічний збиток. Транспорт, що простоює в заторах або рухається в режимі частих прискорень та гальмувань, викидає в атмосферу значно більшу кількість шкідливих речовин (CO_2 , NO_x , тверді частки). Це погіршує якість повітря в містах, сприяє зміні клімату та негативно впливає на здоров'я населення, підвищуючи ризик респіраторних та серцево-судинних захворювань. Окрім того, підвищується рівень шумового забруднення, що знижує комфорт проживання в міських районах.

Соціальні проблеми. Збільшення часу в дорозі викликає стрес та втому у водіїв та пасажирів, знижуючи якість життя. Непередбачуваність тривалості поїздок ускладнює планування дня. Також, перевантажені перехрестя можуть створювати небезпечні ситуації та підвищувати ризик дорожньо-транспортних пригод.

Постановка задачі полягає в тому, щоб створити автономний алгоритм керування світлофорами, який би ефективно використовував дані, зібрані за допомогою вищезгаданої «типової» системи (камера, ПК, контролер), для мінімізації негативних наслідків заторів. Зокрема, планується застосування методів підкріплювального навчання – Q-learning, DQN (Deep Q-Network) і SARSA – для прийняття рішень про зміну світлофорних фаз у реальному часі. Q-learning і SARSA, як класичні алгоритми RL, оперують табличними представленнями станів

і дій та оновлюють свою політику управління на основі досвіду взаємодії з моделюванням дорожнього руху. DQN, натомість, використовує глибоку нейронну мережу для апроксимації функції цінності (Q-function), що дозволяє працювати з більш складними, високорозмірними або навіть неперервними просторами станів, що є характерним для реальних дорожніх умов.

Таким чином, проблема звужується до інтеграції стандартних компонентів апаратної інфраструктури з інтелектуальними алгоритмами навчання з підкріпленням для забезпечення повністю автономного та адаптивного регулювання дорожнього руху на окремому перехресті. Передбачається, що агент, реалізований переважно на мові Python, буде навчатися на даних, отриманих із симуляційного середовища (наприклад, SUMO) або, в перспективі, з реального трафіку, і потім передаватиме керуючі команди дорожньому контролеру через мережевий інтерфейс.

Основним викликом є забезпечення безпечної, надійної та ефективної поведінки розробленого алгоритму в широкому діапазоні дорожніх умов, включаючи нетипові ситуації, такі як дорожньо-транспортні пригоди, перекриття смуг руху або різкі непередбачувані зміни в інтенсивності трафіку. Табличні алгоритми, такі як SARSA та Q-learning, можуть суттєво покращити ситуацію (наприклад, скоротити довжину черг на перехрестях до 19%, як показано на екстремальних прикладах на рис. 1.7), але їхня ефективність може бути обмежена через необхідність дискретизації простору станів, що не завжди адекватно відображає складність реальних дорожніх сценаріїв.



Рисунок 1.7 – Екстремальний приклад черг на перехрестях

Джерело: [12]

Для створення дійсно ефективної системи необхідно не тільки порівняти декілька підходів навчання з підкріпленням, але й розглянути можливість їх гібридизації або поєднання з класичними методами управління трафіком для забезпечення більшої відмовостійкості та передбачуваності поведінки системи, особливо в критичних ситуаціях.

Успішне вирішення цього завдання призведе до низки позитивних результатів:

- **зменшення затримок та підвищення пропускної здатності перехресть.** Оптимізоване управління світлофорами дозволить скоротити час очікування для всіх учасників руху, що призведе до більш плавного та швидкого проходження перехресть;
- **покращення економічної ефективності.** Зменшення витрат пального та скорочення часу в дорозі матиме прямий позитивний економічний ефект;
- **зниження негативного впливу на довкілля.** Менше часу роботи двигунів на холостому ходу та більш плавний рух сприятимуть зменшенню викидів шкідливих речовин;

- **підвищення безпеки дорожнього руху.** Зменшення заторів та більш передбачувана робота світлофорів можуть сприяти зниженню кількості ДТП;
- **підвищення комфорту для учасників дорожнього руху.**
- Скорочення часу в дорозі та зменшення стресу позитивно вплинуть на загальне самопочуття людей.

Кінцева мета полягає в розробці алгоритмів, які, спираючись на існуючу або типову апаратну базу (за прикладом обладнання SEA), зможуть забезпечити ефективний автономний режим керування світлофорами. Це дозволить досягти мінімальних затримок транспортних засобів, підвищити пропускний потенціал перехресть та, як наслідок, покращити якість життя в міському середовищі.

1.4 Обґрунтування вибору підходів і технологій для створення СШІ

Для розробки й експериментів обрано мову програмування **Python**, оскільки вона має багатий набір бібліотек для машинного навчання та симуляції (рис 1.8) [13].

Зокрема, існує ціла екосистема (TensorFlow, PyTorch, Stable Baselines, Rllib) для швидкого прототипування моделей RL.



Рисунок 1.8 – Бібліотеки для машинного навчання

Для реалістичної симуляції трафіку під час тренування моделей буде використовуватися середовище SUMO доступне з Python-інтерфейсом TraCI (Traffic Control Interface), що полегшує підключення до симуляцій дорожнього руху [15].

Для аналізу даних, візуалізації результатів та побудови графіків у рамках розробки системи було обрано інструмент JupyterLite – легку версію Jupyter Notebook, яка працює повністю в браузері без необхідності встановлення додаткового програмного забезпечення [16]. JupyterLite базується на WebAssembly та дозволяє запускати Python-код прямо в браузері, що робить його ідеальним рішенням для мобільної розробки, тестування та демонстрацій (рис. 1.9).

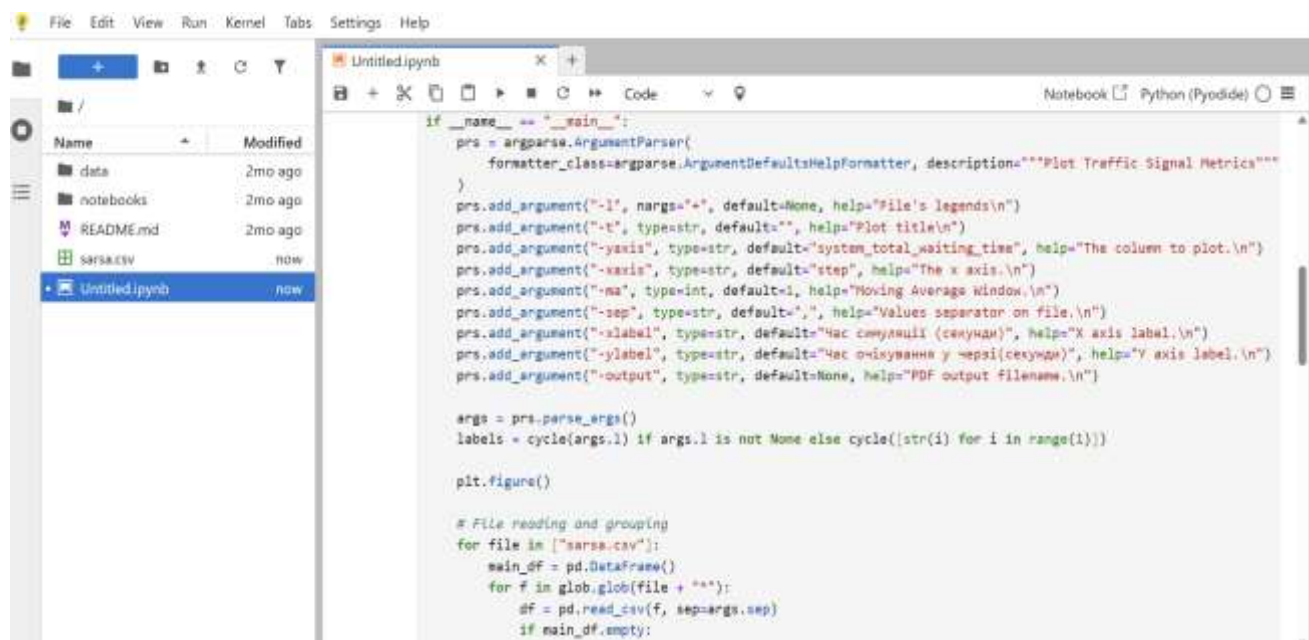


Рисунок 1.9 – Середовище JupyterLite

Джерело: складено самостійно

Оскільки після завершення тренування моделей результати не потребують великих обчислювальних ресурсів, аналіз можна ефективно виконувати без використання потужного «заліза». Завдяки цьому для вивчення поведінки агента, обробки логів та побудови графіків можна застосовувати браузерні інструменти, такі як JupyterLite, що значно спрощує розгортання та повторюваність експериментів навіть у слабких або обмежених середовищах.

Eclipse SUMO (Simulation of Urban MObility) – це мікроскопічна платформа з відкритим вихідним кодом для моделювання дорожнього руху, яка розробляється в рамках проєкту Eclipse (рис. 1.10). Вона дозволяє точно симулювати динаміку індивідуальних транспортних засобів, пішоходів, велосипедистів, громадського транспорту та світлофорів [17]. Серед її ключових можливостей - підтримка великомасштабного моделювання, обробка реальних дорожніх мереж (імпорт з OpenStreetMap), а також висока продуктивність навіть при роботі з великими сценаріями.

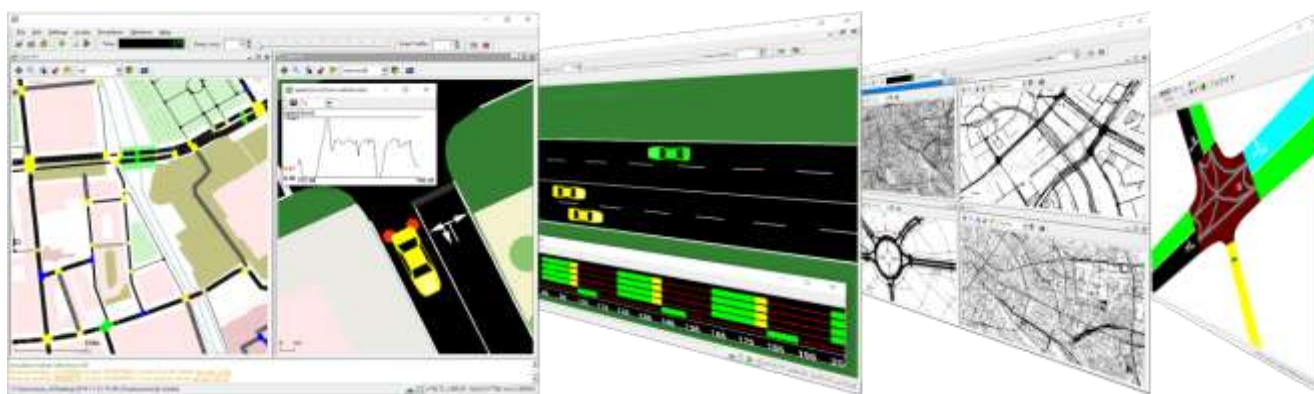


Рисунок 1.10 – Можливості інтерфейсу SUMO

Джерело: [17]

SUMO підтримує різні режими управління трафіком: фіксовані цикли світлофорів, адаптивні алгоритми, а також зовнішнє керування через TraCI (Traffic Control Interface) – потужний інтерфейс, що дозволяє інтегрувати власні алгоритми на Python, C++ чи інші мови.

SUMO добре підходить для розроблення систем автономного управління рухом на перехрестях:

- дає змогу точно моделювати поведінку транспортних засобів на перехрестях, включно з чергами, затримками та реакціями на сигнали;
- завдяки TraCI можна в реальному часі змінювати фази світлофорів, додавати/видаляти транспортні одиниці та отримувати детальні дані про трафік;

- підтримує сценарії з інтелектуальними агентами, що робить її придатною для тестування систем штучного інтелекту й машинного навчання;
- дає можливість моделювати змішані сценарії – як із людським керуванням, так і з автономними транспортними засобами;
- детальна візуалізація та логування дозволяють легко аналізувати ефективність алгоритмів керування.

Завдяки цьому SUMO є одним із найпопулярніших інструментів серед дослідників і розробників систем інтелектуального транспортного управління, зокрема на перехрестях.

Водночас для демонстраційних і спрощених симуляційних моделей може бути корисною **TypeScript**. За його допомогою можна створити браузерну візуалізацію роботи агентів або інтерактивний прототип контролера (рис. 1.11). Наприклад, фреймворки на основі TensorFlow.js дозволяють реалізувати прості нейромережі у веб-середовищі. Це не основне рішення для кінцевих тестів, але дає можливість проілюструвати принцип автономного керування світлофорами без встановлення складного ПЗ.



Рисунок 1.11 – Можливості візуалізації

Джерело: [18]

Для розробки візуалізацій, симуляційних компонентів та допоміжних скриптів використовувалось середовище Visual Studio Code (VS Code), яке

забезпечує зручний та продуктивний процес розробки завдяки широкому набору розширень та інтеграцій з інструментами веб- і Python-екосистем [19].

Зокрема, для прискорення написання рутинного коду, а також генерації шаблонних функцій активно використовувався інструмент GitHub Copilot [20] (рис. 1.12). Його застосування дало змогу значно скоротити час на прототипування та автоматизувати повторювані дії, зберігаючи при цьому контроль над логікою реалізації.

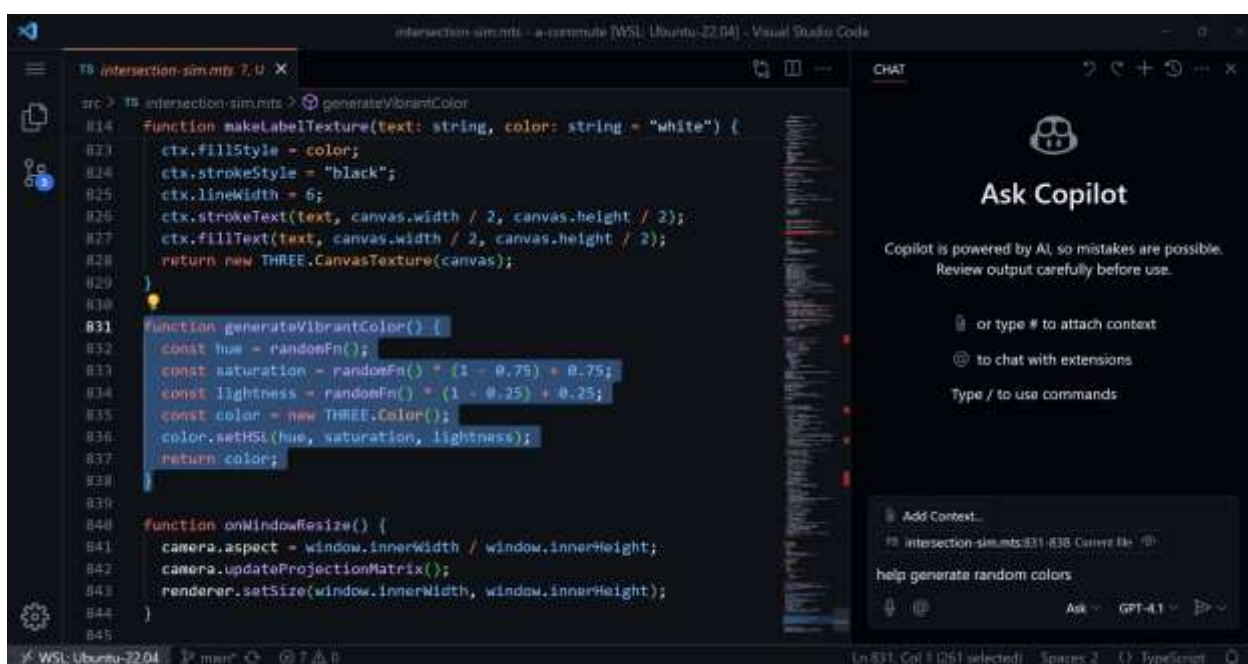


Рисунок 1.12 – Середовище VS Code та приклад використання Copilot

Джерело: розроблено автором

VS Code також слугував основною платформою для одночасної роботи з кількома мовами програмування – Python, TypeScript і HTML/CSS, що було важливо для створення узгодженого стеку інструментів для бекенду (тренування моделей) та фронтенду (візуалізація результатів).

Що ж до алгоритмів, то **Q-learning**, **DQN** і **SARSA** обрано через їх визнання у вирішенні завдань інтелектуальних транспортних систем. Q-learning та SARSA є найбільш класичними підходами RL з табличним представленням функції якості; вони забезпечують зрозумілий базовий рівень адаптивності агента. DQN розширює

цю парадигму, вводячи глибокі нейромережі для апроксимації Q-функції – робить можливим працювати з великими просторами станів (наприклад, безперервними показниками щільності трафіку). Як показано в оглядах, описаних вище, ці методи вже довели свою ефективність у контексті керування трафіком: RL-алгоритми дозволили значно зменшити час простою автомобілів на перехресті. Таким чином, комбінуючи Python і TypeScript (TensorFlow.js) для реалізації та демонстрації агентів, а також використовуючи Q-learning/DQN/SARSA для навчання контролера, забезпечується комплексний підхід до розробки автономної системи.

Висновки до Розділу 1

У першому розділі проведено ґрунтовний аналіз предметної області автономного управління дорожнім рухом на перехрестях.

Зроблено огляд теоретичних основ та наголошено на актуальності адаптивного регулювання світлофорів в умовах міського трафіку. Розглянуто можливості існуючого обладнання (камери, ПК, контролери SEA) і підтверджено, що воно здатне підтримувати розгортання інтелектуальних алгоритмів на практиці. Аналіз існуючих рішень показав, що сучасні системи штучного інтелекту (глибоке навчання, підкріплювальне навчання) ефективно вирішують допоміжні задачі (наприклад, детекцію транспорту або адаптацію сигналів в режимі моделювання). При цьому виявлено відсутність квазі-реальних впроваджень повністю автономного регулювання на базі стандартного обладнання.

Визначено, що для досягнення автономності у реальному часі доцільно поєднати існуючі технології: Python-симуляції в SUMO, глибинні та табличні алгоритми RL і, за потреби, легкі веб-інтерфейсні реалізації. Отримані висновки дозволили чітко сформулювати постановку завдань та виправдати вибір підходів (Python, TypeScript, Q-learning, DQN, SARSA) для подальшої реалізації системи автономного управління світлофорами.

РОЗДІЛ 2

ХАРАКТЕРИСТИКА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ ТА ПОСТАНОВКА ЗАДАЧ

2.1 Характеристика об'єкта дослідження

Системи управління дорожнім рухом регулюють переміщення транспортних засобів і пішоходів на дорогах та перехрестях, забезпечуючи безпеку та ефективність руху. Вони охоплюють як автоматизовані засоби – світлофори, регулятори в'їзду на автомагістралі, табло зі змінною швидкістю – так і статичні елементи, як дорожні знаки та розмітка. У цій роботі під терміном «система управління рухом» маються на увазі саме світлофорні об'єкти, встановлені на перехрестях.

Щоб краще розуміти системи управління дорожнім рухом, спершу потрібно ознайомитись із базовими поняттями [9]:

Перехрестя. Перехрестя – це місце, де перетинаються або з'єднуються дві чи більше доріг, дозволяючи транспортним засобам і пішоходам змінювати напрямки руху. Найбільш типовими перехрещеннями є прямокутні, симетричні, несиметричні, У- і Т- подібні тощо [21] (рис 2.1).

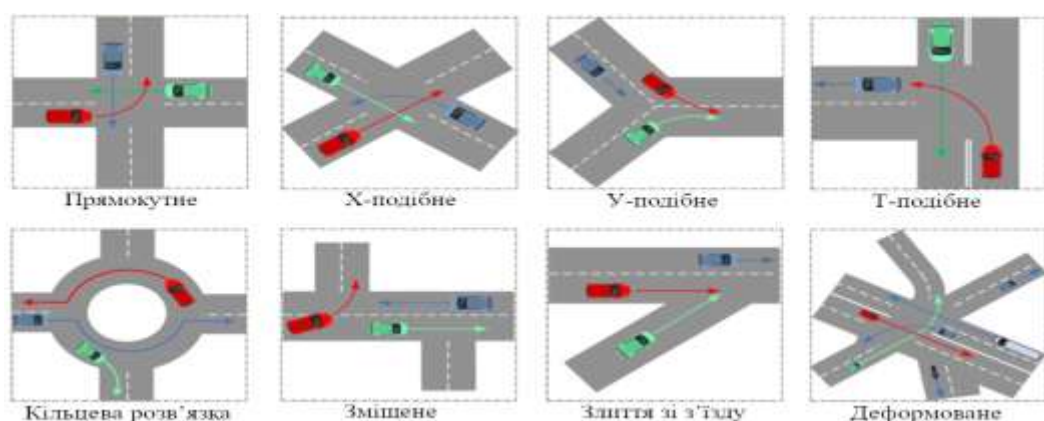


Рисунок 2.1 – Типи перехресть

Джерело: розроблено автором на основі [21]

Підхід. Підходом називають смугу або дорогу, що веде до перехрестя. Це траєкторія, якою рухається транспорт у напрямку до місця перетину.

Смуга руху. Смуга – це виділена частина дороги, призначена для руху одного ряду транспортних засобів. Смуги часто позначаються дорожньою розміткою і можуть мати спеціальні призначення, наприклад, для повороту, злиття потоків або прямого руху.

Маневр. Маневр – це напрям або траєкторія, яку обирають транспортні засоби чи пішоходи під час проходження перехрестя. Маневри поділяються за напрямком: прямий рух, лівий поворот, правий поворот або пішохідний перехід. У цій роботі розглядаються переважно маневри транспортних засобів.

Фаза. Фаза у світлофорному регулюванні – це період, протягом якого певним транспортним потокам або пішоходам дозволено рухатися через перехрестя. Кожна фаза відповідає окремому маневру і визначається конкретною комбінацією сигналів.

Групи фаз. Це набори фаз, які можуть виконуватись одночасно без конфліктів

Послідовність фаз. Це визначений порядок активації фаз на перехресті. У фазах однієї групи сигнали можуть працювати одночасно. Зазвичай послідовність є фіксованою, але її можна змінювати.

Цикл. Повний період, протягом якого активуються всі фази на перехресті. Включає зелений, жовтий і червоний сигнали. Вимірюється в секундах.

Зсув. Часова різниця між початком зеленої фази на одному перехресті та відповідною фазою на іншому. Використовується для синхронізації світлофорів.

Розподіл часу. Частки часу циклу, виділені на кожен фазу. Визначаються з урахуванням трафіку і виражаються у відсотках від загального часу циклу.

Зелений час. Проміжок, протягом якого горить зелений сигнал, дозволяючи рух.

Червоний час. Період дії червоного сигналу, коли рух заборонений.

Жовтий час. Перехідна фаза між зеленим і червоним, що попереджає водіїв про необхідність зупинки.

Мінімальний зелений час. Найкоротша тривалість зеленого сигналу, що дозволяє транспортним засобам розпочати рух і проїхати перехрестя.

Максимальний зелений час. Найдовша допустима тривалість зеленої фази, щоб уникнути затримок для інших потоків і забезпечити справедливий розподіл часу.

Робота світлофорів полягає в регулюванні руху на перехрестях через послідовність фаз, кожна з яких керує окремими маневрами. Фази організовані у фіксовану послідовність, яка повторюється протягом циклу. Протягом зеленої фази рух дозволений, а під час червоної – заборонений (рис. 2.2).

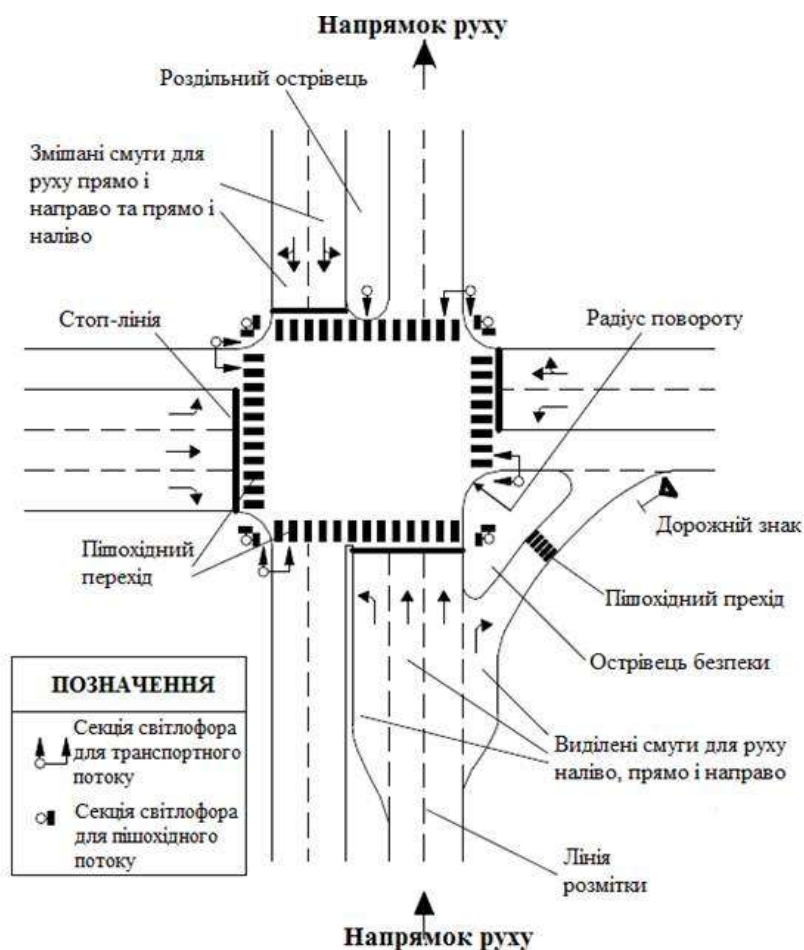


Рисунок 2.2 – Детальна схема регульованого перехрестя та елементів управління транспортними та пішохідними потоками

Джерело: [22]

Системи управління рухом можна класифікувати за такими категоріями:

- керування;
- координація;
- таймінг.

Приклад класифікації подано на рис. 2.3.

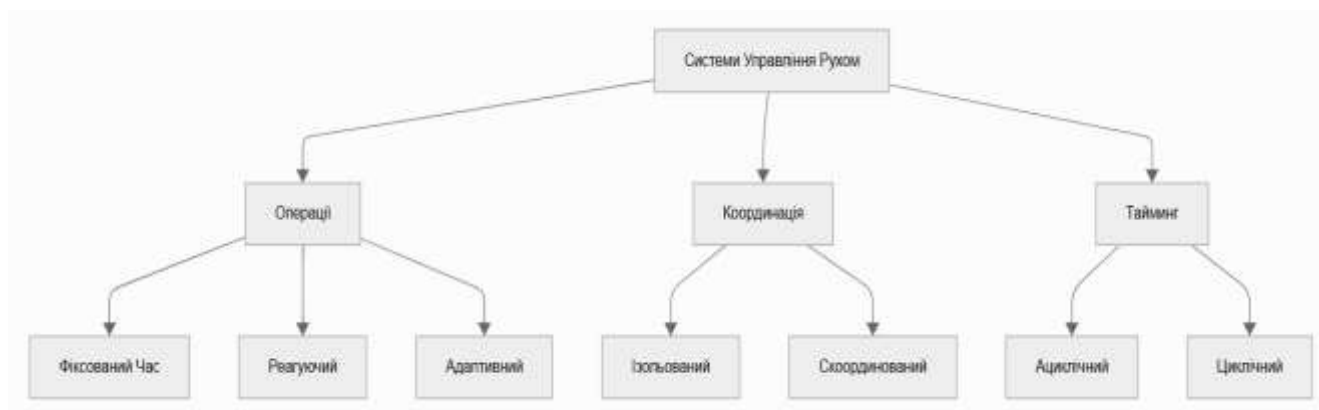


Рисунок 2.3 – Класифікація систем управління рухом

Джерело: розроблено автором

З часом було впроваджено багато систем управління дорожнім рухом. Загалом їх можна поділити на три основні категорії за принципом роботи:

- фіксоване таймерне керування;
- реагуюче керування;
- адаптивне керування.

Фіксоване таймерне керування. Перші системи управління дорожнім рухом працювали вручну та контролювались поліцейськими. Навіть сьогодні в багатьох країнах, що розвиваються, регулювання руху досі здійснюється вручну. Із розвитком технологій з'явилися автоматизовані системи управління рухом з фіксованим розкладом сигналів [23]. Це найпоширеніший тип систем у світі. Світлофори з фіксованим таймінгом працюють за попередньо заданими інтервалами, що повторюються циклічно. Іноді передбачена можливість зміни розкладу залежно від часу доби. Загалом, такі системи не мають зворотного зв'язку і не адаптуються до змін у дорожньому русі.

Реагуюче керування. Згодом у розвитку світлофорних систем з'явилося реагуюче (керування [24]). На відміну від систем із фіксованим таймінгом, воно реагує на зміни дорожнього руху в реальному часі. Це досягається за допомогою детекторів, інтегрованих зі світлофорним контролером.

Оскільки система базується на виявленні транспортних засобів на підходах до перехрестя, для її проектування використовуються специфічні параметри. Мінімальний зелений час – це найкоротший період, протягом якого зелений сигнал має залишатися увімкненим, щоб хоча б один автомобіль міг проїхати перехрестя. Якщо транспорт присутній, зелений сигнал може бути продовжений, але не довше за максимальний зелений час – граничну тривалість зеленої фази. У таких системах немає фіксованого циклу, як у випадку зі світлофорами на фіксованому таймері.

Адаптивне керування. З появою потужного обчислювального обладнання були впроваджені адаптивні системи управління дорожнім рухом (ATCS) [25], також відомі як системи міського управління рухом. Це більш розвинена форма реагуючого керування. Як і в реагуючих системах, адаптивні системи використовують сенсори для збору даних і керування світлофорами. Проте на відміну від них, ATCS постійно адаптується до змін у русі та оновлює налаштування контролерів у режимі реального часу. Такі системи не покладаються на заздалегідь задані параметри, а оновлюють їх на основі даних з реального трафіку.

Найкраще розглядати ці три типи систем як інструменти, а не як історію еволюції. Залежно від місця впровадження, бюджетних обмежень та інших вимог, оптимальною може бути будь яка із систем. Але ця робота зосереджена на ATCS, адже автоматизація руху розглядається саме в контексті міського середовища.

2.2 Структура і характеристика СШІ

Система штучного інтелекту (СШІ) для автономного управління дорожнім рухом на перехрестях складається з ряду функціональних компонентів, об'єднаних у логічну структуру, яка дозволяє здійснювати моніторинг, аналіз дорожньої ситуації та керування світлофорами на основі реального часу. Загальну діаграму системи можна побачити на рис. 2.4.

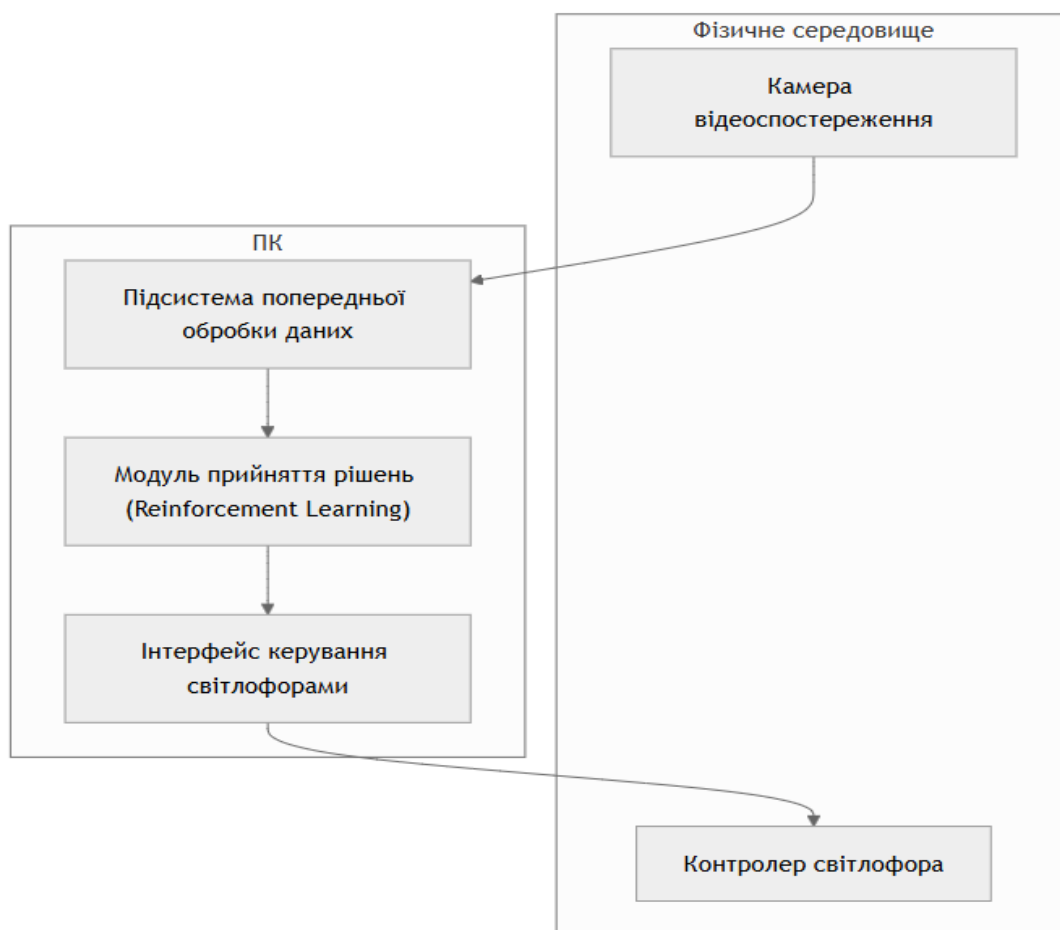


Рисунок 2.4 – Компоненти системи

Джерело: розроблено автором

Сенсорна підсистема. Отримує вхідні дані за допомогою відеокамер, розміщених на перехресті. Камери забезпечують постійний відеопотік, який використовується для виявлення транспортних засобів, їх кількості, напрямку та швидкості.

Підсистема попередньої обробки даних. Включає модулі комп'ютерного зору (наприклад, YOLO, OpenCV) для обробки відео в реальному часі, фільтрацію шумів, визначення ключових об'єктів (автомобілів, пішоходів), перетворення необроблених відеоданих у структуровану інформацію.

Модуль прийняття рішень. Є ядром СШІ та фокусом розробки в цій роботі. Модуль базується на методах підкріплювального навчання (Reinforcement Learning), що дозволяє системі динамічно адаптувати фази світлофорного регулювання під реальний трафік. Агент навчається шляхом спостереження за ефектами своїх дій (наприклад, зменшення черг) і поступово оптимізує політику керування.

Виконавча підсистема. Перетворює вихідні дані з модуля прийняття рішень у сигнали, які надсилаються контролеру світлофорів. Інтерфейс взаємодії з контролером реалізується через стандартні цифрові порти ПК або протоколи, підтримувані обладнанням.

Архітектурний стиль.

Оскільки система реалізується на вже існуючому фізичному обладнанні – одному персональному комп'ютері, контролері світлофорів і камері – було обрано монолітну архітектуру з модульною організацією коду. Усі компоненти розгортаються на одному ПК, комунікація між модулями відбувається через внутрішні виклики в межах одного процесу, що спрощує розгортання та знижує вимоги до ресурсів.

Слід враховувати, що існуюче програмне забезпечення, встановлене на обладнанні керування світлофорами, має можливість централізованого оновлення з контрольного пункту. У зв'язку з цим розроблювана система повинна бути сумісною з поточним циклом керування та інтегруватися без порушення основної логіки роботи але через те, що використовуване ПЗ є пропрієтарним, його технічні

специфікації та API залишаються невідомими. Для гарантій безпеки та безперервності роботи СШ передбачається резервний режим управління, який автоматично активується у разі відмови основного ШІ-модуля. У цьому режимі управління світлофорами здійснюється за фіксованими таймерами без участі інтелектуальних алгоритмів.

2.3 Принципи та алгоритми для оптимізації дорожнього руху

Для побудови системи автономного управління дорожнім рухом на перехрестях у цій роботі застосовуються методи підкріплювального навчання, що є ефективним підходом до оптимізації рішень у динамічному середовищі з неповною інформацією [26].

У моделі підкріплювального навчання визначаються три ключові компоненти (рис. 2.5):

State (s) – стан системи, що описує ситуацію на перехресті, наприклад, кількість автомобілів у черзі на кожному напрямку.

Action (a) – дія, яку може виконати агент, тобто зміна фаз світлофора (перемикання зеленого/червоного).

Reward (r) – винагорода, що визначає корисність дії агента; наприклад, зменшення довжини черг або часу очікування.



Рисунок 2.5 – Компоненти моделі підкріплювального навчання

Джерело: розроблено автором

Мета агента – навчитися політики (policy), яка максимізує сумарну очікувану винагороду в довгостроковій перспективі.

Для реалізації RL у межах цієї роботи розглядаються наступні алгоритми:

- Q-навчання (Q-learning);
- SARSA;
- DQN.

Q-навчання. Q-learning – це алгоритм навчання з підкріпленням, який дозволяє агенту оптимізувати свою стратегію дій у динамічному середовищі, прагнучи максимізувати суму майбутніх винагород. Агент досліджує середовище, приймаючи рішення на основі попереднього досвіду, а не за попередньо визначеною моделлю світу [27]. Ядром алгоритму є рівняння Беллмана [формула (2.1)] як просте уточнення з ітерацією за цінностями, що використовує зважене усереднення старої цінності та нової інформації [28]:

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times \left(r_t + \gamma \times \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right), \quad (2.1)$$

де $Q(s_t, a_t)$ – старе значення, α – коефіцієнт навчання, r_t – винагорода, γ – коефіцієнт дисконтування, $\max_a Q(s_{t+1}, a)$ – оцінка оптимального майбутнього значення

Джерело: [16]

Політика ϵ -жадібності допомагає агентові вирішити, яку дію виконати на основі поточних оцінок Q-значень:

- **експлуатація:** з імовірністю $1-\epsilon$ агент обирає дію з найвищим Q-значенням. Це означає використання наявних знань для максимізації винагороди.
- **дослідження:** з імовірністю ϵ агент обирає випадкову дію, досліджуючи нові можливості, щоб з часом знайти кращі стратегії й покращити прийняття рішень.

SARSA. Алгоритм SARSA (State Action Reward State` Action`) трохи відрізняється від Q-learning. У алгоритмі SARSA значення Q оновлюється з

урахуванням дії a_1 , виконаної в стані s_1 [29]. У Q-learning для оновлення Q-таблиці використовується дія з найвищим значенням Q у наступному стані s_1 [формула (2.2)].

$$Q^{new}(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \times (r_t + \gamma \times Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)), \quad (2.2)$$

де $Q(s_t, a_t)$ – старе значення, α – коефіцієнт навчання, r_t – винагорода, γ – коефіцієнт дисконтування, $Q(s_{t+1}, a_{t+1})$ – наступний стан

Джерело: [17]

За алгоритмом SARSA, агент взаємодіє з середовищем та оновлює стратегію згідно з виконаними діями, отже, цей алгоритм можна віднести до класу алгоритмів навчання за поточною стратегією (on-policy). Значення Q-функції для дії та стану оновлюється відповідно похибці, що регулюється за допомогою коефіцієнту швидкості навчання α . Значення Q-функції представляє сумарну винагороду, яку можна отримати за весь час, що залишився у межах цього епізоду, за умови виконання дії a в стані s , з додаванням знеціненої винагороди за виконання дії в наступному стані.

Процес SARSA починається з ініціалізації $Q(s_t, a_t)$ довільними значеннями. На цьому етапі встановлюється початковий поточний стан (S), і початкова дія (A) вибирається за допомогою політики ϵ -жадібного алгоритму на основі поточних значень Q. Політика ϵ -жадібності поєднує використання експлуатації та дослідження в процесі навчання для вибору дії з найвищою оцінкою винагороди.

Експлуатація полягає у використанні вже відомих, оцінених значень для отримання більшої кількості винагород, отриманих раніше. Дослідження передбачає спроби знайти нові знання про дії, що може призвести до короткострокових неоптимальних рішень під час навчання, але згодом дати змогу знайти найкращу можливу дію і винагороду.

DQN. Традиційні методи RL, такі як Q-learning і методи градієнта політики, покладаються на табличні представлення або лінійні апроксимації, що часто не масштабуються до високовимірних або неперервних просторових входів.

Один із основних недоліків Q-навчання полягає в тому, що воно стає непридатним при роботі з великими просторами станів, оскільки розмір Q-таблиці зростає експоненційно з кількістю станів і дій. У таких випадках алгоритм стає обчислювально дорогим і потребує багато пам'яті для зберігання Q-значень. Уявімо гру з 1000 станів і 1000 дій на кожен стан – нам знадобиться таблиця з мільйоном комірок. І це ще дуже малий простір станів у порівнянні з шахами або Го. Крім того, Q-навчання не можна використовувати для невідомих станів, оскільки воно не може передбачити Q-значення нових станів на основі попередніх [30].

Це створює дві проблеми:

- обсяг пам'яті, необхідний для збереження та оновлення таблиці, збільшується зі зростанням кількості станів.
- час, необхідний для дослідження кожного стану з метою побудови необхідної Q-таблиці, стає нереалістичним.

Щоб подолати цю проблему, альтернативним підходом є поєднання Q-навчання з глибинними нейронними мережами. Цей підхід називається глибинним Q-навчанням (Deep Q-Learning, DQL). Нейронні мережі в DQL виступають як апроксиматори Q-значень для кожної пари (стан, дія).

Нейронна мережа отримує стан на вхід і видає Q-значення для всіх можливих дій. Наступна ілюстрація (рис. 2.6) показує різницю між Q-навчанням і глибинним Q-навчанням у оцінюванні Q-значень.

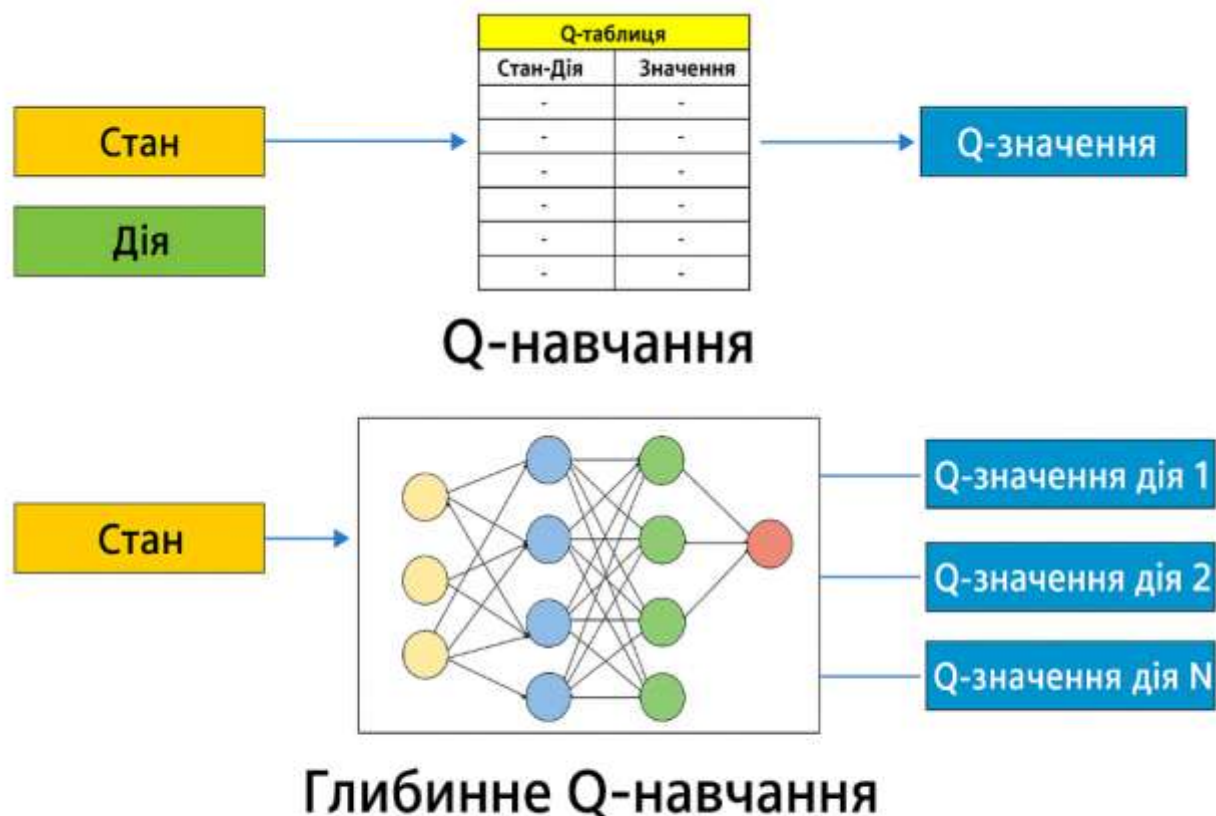


Рисунок 2.6 – Порівняння підходів Q-навчання та глибинного Q-навчання

Джерело: розроблено автором на основі [31]

Основно, глибоке Q-навчання замінює звичайну Q-таблицю нейронною мережею. Замість того, щоб зв'язувати пару (стан, дія) з Q-значенням, нейронна мережа відображає вхідні стани в пари (дія, Q-значення).

У 2013 році компанія DeepMind представила алгоритм Deep Q-Network (DQN) [32]. DQN призначений для навчання грати в ігри Atari з сирих пікселів. Це стало проривом у галузі підкріплювального навчання та допомогло прокласти шлях для майбутніх досягнень у цій області. Термін "Deep Q-network" стосується нейронної мережі в архітектурі глибинного Q-навчання.

У процесі розробки системи буде проведено порівняння ефективності вищезазначених алгоритмів для визначення найбільш оптимального підходу до управління дорожнім рухом на перехрестях.

Висновки до Розділу 2

У Розділі 2 було проведено характеристику об'єкта дослідження та постановку задачі в контексті автономного управління дорожнім рухом на перехрестях за допомогою систем штучного інтелекту.

Розглянуто ключові поняття, що стосуються систем управління дорожнім рухом, зокрема світлофорних об'єктів на перехрестях, визначено такі елементи як перехрестя, підхід, смуга руху, маневр, фаза, цикл та інші. Обґрунтовано вибір адаптивних систем управління дорожнім рухом як основного напрямку дослідження, зважаючи на їхню здатність до постійної адаптації до реальних умов трафіку в міському середовищі.

Детально описано структуру запропонованої системи штучного інтелекту для автономного управління та виконавчої підсистеми. Визначено, що архітектурний стиль системи є монолітним з модульною організацією, що зумовлено використанням існуючого фізичного обладнання. Окреслені питання інтеграції з пропрієтарним програмним забезпеченням світлофорних контролерів та передбачено резервний режим роботи на основі фіксованих таймерів для забезпечення безперервності та безпеки управління.

Представлено методи, моделі та алгоритми, що будуть застосовуватися для розробки модуля прийняття рішень – методи підкріплювального навчання. Розглянуто три потенційні алгоритми для реалізації: Q-навчання, SARSA та глибинне Q-навчання (DQN). Проаналізовано їхні принципи функціонування, зокрема переваги DQN для роботи з великими просторами станів за допомогою нейронних мереж.

Таким чином, у цьому розділі закладено фундаментальні основи для подальшої роботи: визначено об'єкт дослідження, його характеристики, структуру проєктованої СШІ та обрано базовий методологічний підхід (підкріплювальне навчання) з окресленням конкретних алгоритмів, які будуть досліджуватися та порівнюватися для досягнення мети роботи.

РОЗДІЛ 3

РОЗРОБЛЕННЯ ПРОЄКТНИХ РІШЕНЬ

3.1 Загальний підхід до створення системи

Проектування ефективної системи автономного управління дорожнім рухом на перехрестях є багатоетапним процесом, що вимагає ретельного планування та послідовного виконання ключових завдань. Основою успіху є комплексний підхід, що охоплює як підготовку даних та вибір моделей, так і їх ретельне навчання та тестування.

Перш за все, критично важливим є забезпечення якісних вхідних даних. Ці дані повинні максимально точно та повно відображати майбутні умови роботи системи, включаючи різноманітні сценарії дорожнього руху: пікові години, періоди низького завантаження, вплив погодних умов (якщо це враховується), а також можливі непередбачувані події, такі як дорожні роботи чи ДТП. Якість даних безпосередньо впливає на здатність моделі навчатися адекватним стратегіям управління. Неповні або спотворені дані можуть призвести до розробки системи, що буде неефективною або навіть контрпродуктивною в реальних умовах. Тому збір, очищення та попередня обробка даних є фундаментальним етапом.

Далі, не менш важливим кроком є обрання відповідних моделей представлення станів, дій та винагород. Оскільки система розробляється на основі навчання з підкріпленням (Reinforcement Learning, RL), ці три компоненти є серцевиною RL-агента:

- **стан (state).** Це формалізований опис поточного середовища, який агент отримує для прийняття рішення. У контексті управління світлофором стан може включати інформацію про поточну фазу світлофора, кількість транспортних засобів на під'їзних смугах, довжину черг, час очікування першого автомобіля в черзі, час, що минув з моменту останньої зміни фази тощо. Точне та інформативне

представлення стану є ключовим для того, щоб агент міг розрізняти ситуації, що вимагають різних управлінських рішень;

- **дія (action):** Це набір можливих операцій, які агент може виконати в кожному стані. Для світлофорного агента діями можуть бути перемикання на наступну визначену фазу, продовження поточної зеленої фази на певний час, або вибір однієї з кількох можливих фаз для активації. Простір дій має бути достатньо гнучким, щоб дозволяти ефективне управління, але не надто великим, щоб уникнути надмірної складності навчання;

- **винагорода (reward):** Це числовий сигнал, який агент отримує від середовища після виконання дії у певному стані. Винагорода вказує на те, наскільки добре була виконана дія з точки зору досягнення поставленої мети (наприклад, мінімізації загального часу очікування, максимізації пропускну здатності). Проектування функції винагороди є одним з найважливіших та найскладніших аспектів RL, оскільки вона безпосередньо формує поведінку агента.

На заключному етапі розробки проектних рішень слід провести навчання та тестування кількох підходів до RL. У даній роботі акцент робиться на таких алгоритмах, як Q-learning, DQN та SARSA. Вибір цих алгоритмів зумовлений їхньою різною складністю, підходами до навчання (value-based, on-policy, off-policy) та здатністю працювати з різними типами просторів станів та дій. Це дозволяє провести порівняльний аналіз та визначити найбільш ефективний метод для конкретної задачі.

Моделі порівнюються за низкою критично важливих параметрів:

- **простота реалізації та налаштування.** Алгоритми, що легше імплементуються та вимагають менше зусиль для підбору гіперпараметрів, є більш привабливими для швидкого прототипування та потенційного впровадження;

- **швидкість навчання на доступних обчислювальних потужностях.** Час, необхідний для того, щоб модель досягла прийняттого рівня продуктивності, є важливим фактором, особливо якщо передбачається періодичне перенавчання системи;

- **якість прийняття рішень у симуляції.** Це основний показник ефективності, що оцінюється через метрики, такі як середній час очікування, довжина черг, пропускна здатність перехрестя;

- **стабільність результатів у різних конфігураціях перехресть.** Модель повинна демонструвати надійну роботу не лише на одній конкретній конфігурації, а й бути адаптивною до змін у геометрії перехрестя чи інтенсивності трафіку.

Обраний підхід має забезпечувати оптимальний баланс між продуктивністю та обчислювальною ефективністю. Це особливо актуально в умовах обмежених ресурсів, які можуть бути характерні для вбудованих систем управління трафіком. Надто складна модель може вимагати значних обчислювальних потужностей, що ускладнить її розгортання на реальному обладнанні, тоді як надто проста модель може не забезпечити належної якості управління.

Після ретельної оцінки моделей за вказаними критеріями слід обрати оптимальний алгоритм, який демонструє найкращу адаптацію до різних умов дорожнього руху та відповідає вимогам щодо ефективності та ресурсних обмежень.

Особлива увага при розробці приділяється можливості подальшої інтеграції обраного алгоритму в реальні системи. Це включає декілька аспектів:

- **здатність працювати на обмежених обчислювальних ресурсах.** Багато контролерів світлофорів мають обмежену обчислювальну потужність, тому алгоритм повинен бути достатньо ефективним, щоб виконуватися на такому обладнанні;

- **можливість легкого перенавчання.** Дорожні умови, конфігурації перехресть або інтенсивність трафіку можуть змінюватися з часом (наприклад, через будівництво нових доріг, зміну демографії району). Система повинна легко адаптуватися до таких змін шляхом перенавчання без необхідності повної переробки;

- **стабільна поведінка при нестандартних або аварійних ситуаціях.** Система повинна адекватно реагувати на непередбачені події, такі як несправність датчиків, ДТП на перехресті, або різкі незаплановані зміни у транспортних потоках,

забезпечуючи безпеку та мінімізуючи можливі негативні наслідки. Це може включати перехід до безпечного резервного режиму роботи.

Реалізація цих вимог дозволяє зробити систему максимально універсальною та придатною для практичного впровадження, забезпечуючи її надійність, адаптивність та довгострокову ефективність у динамічному міському середовищі. Такий комплексний підхід формує міцну основу для створення інтелектуальних транспортних систем майбутнього.

3.2 Визначення симуляційного середовища

Для ефективного навчання моделей потрібні дві основні складові:

- конфігурація перехрестя для тренувань і симуляцій;
- якісна модель або запис дорожнього трафіку.

Вибір правильного середовища для симуляцій є критично важливим, оскільки від цього залежить точність моделювання та ефективність навчання. Для створення конфігурації перехрестя важливо враховувати типи та кількість смуг, напрямки руху, можливі перехрестя різних форм та їх взаємозв'язок. Це дозволяє відтворити реалістичні сценарії дорожнього руху та створити умови для ефективної роботи агентів системи.

При дослідженні варіантів було використано OpenStreetMap (OSM) для завантаження даних одного з регіонів Києва та побудови спрощеної симуляції дорожньої мережі [33] (рис. 3.1) (додаток А).

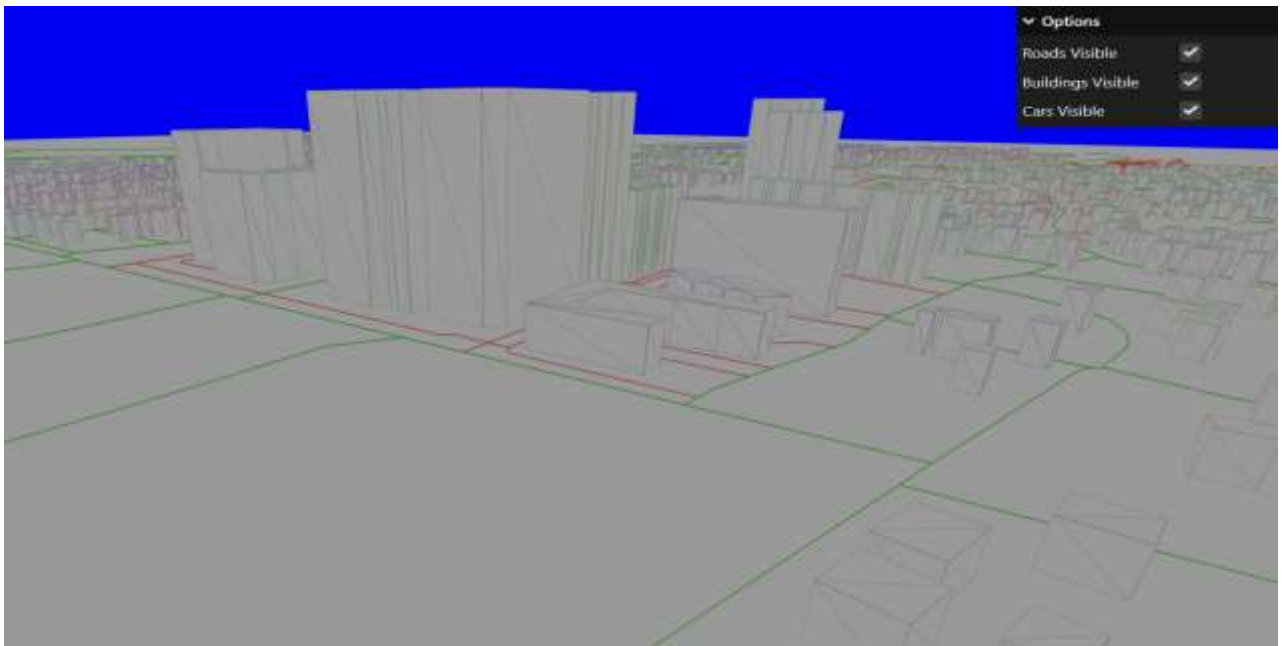


Рисунок 3.1 – Симуляція дорожньої мережі

Джерело: розроблено автором

Також було розроблено просту систему для симуляції трафіку: на карту додається певна кількість авто, кожен з яких отримує випадковий маршрут (рис. 3.2).

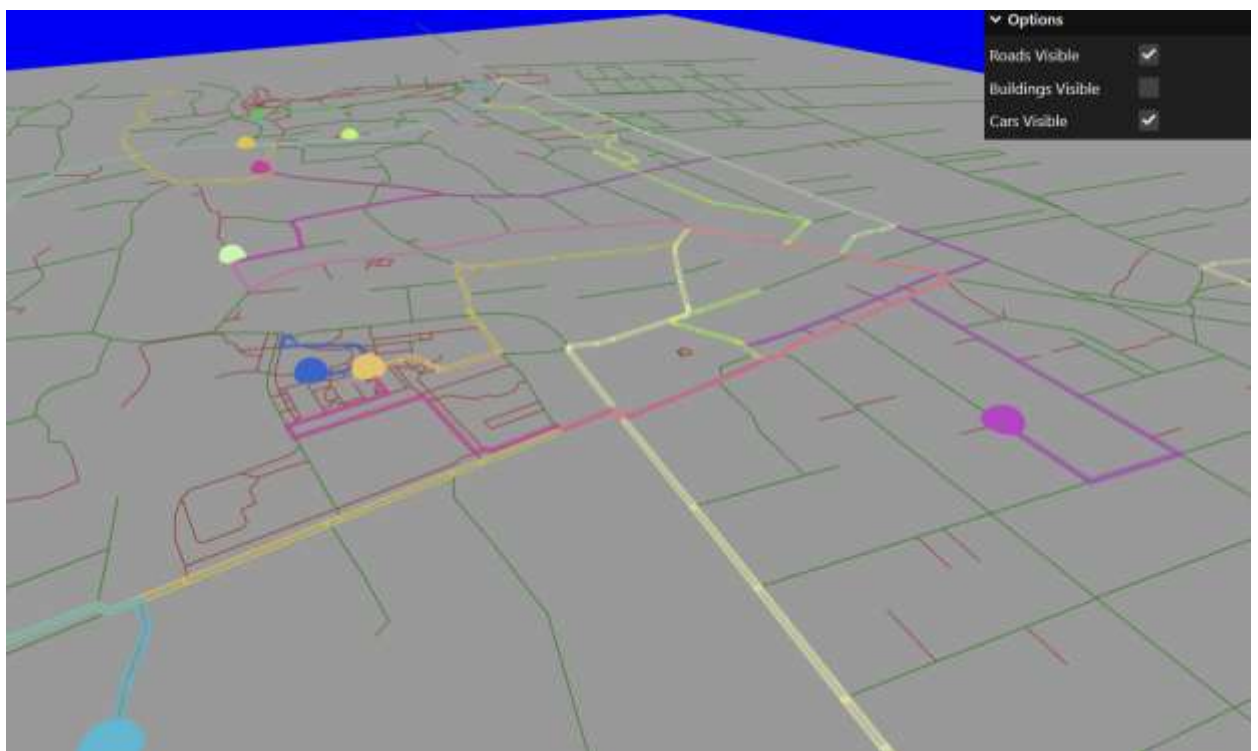


Рисунок 3.2 – Симуляція завантаженості доріг

Джерело: розроблено автором

На жаль ця перша ітерація мала два критичних недоліки:

- так як система була швидко зпрототипована вона не оптимізована для симуляції великої кількості трафіку, що робить її корисною лише для симуляції низької завантаженості доріг але для таких ситуацій достатньо існуючих систем керування дорожнього руху;
- дані отримані з OSM не містять достатньої деталізації перехресть.

Перший недолік можна частково усунути, зробивши фокус на макрорівні симуляції трафіку, симулюючи лише зону перехрестя, зосередившись на ключових параметрах, що впливають на роботу світлофорів. Однак, розробка алгоритмів для реалістичної симуляції трафіку вимагатиме значного часу та зусиль, оскільки це потребує моделювання складних взаємодій між транспортними засобами, пішоходами та світлофорами, а також врахування багатьох факторів, таких як різні типи руху та непередбачувані ситуації.

Другий недолік виявився критичним. На рисунках 3.3 та 3.4 можна побачити приклад перехрестя з деталізацією Google Maps [34].



Рисунок 3.3 – Приклад перехрестя (вид згори)

Джерело: Google Maps



Рисунок 3.4 – Приклад перехрестя (панорамний вигляд)

Джерело: Google Maps

Можна побачити кількість смуг руху, приблизну ширину дороги, наявність пішохідних переходів тощо. Такої деталізації достатньо для побудови акуратної симуляції поведінки транспортного потоку на перехрестях, з урахуванням базових правил дорожнього руху та типових сценаріїв маневрування автомобілів.

Але у випадку з даними з OpenStreetMap деталізація перехрестя виглядає так: (рис. 3.5)

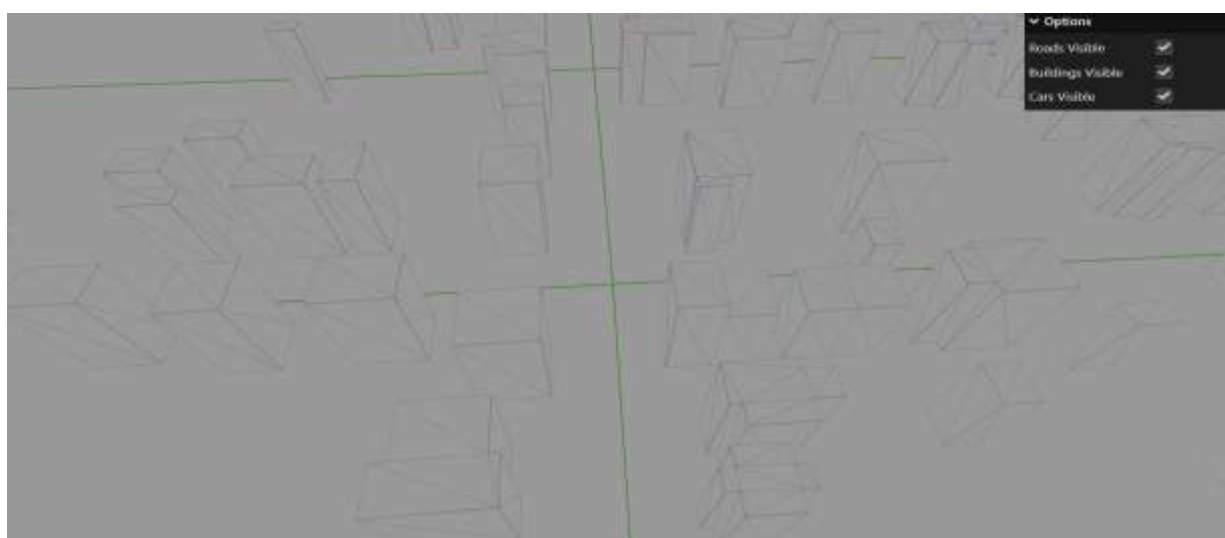


Рисунок 3.5 – Приклад деталізації перехрестя в першому прототипі

Джерело: розроблено автором

Попри наявність деяких базових даних, таких як кількість смуг руху, тип покриття та ширина дороги, цього недостатньо для повноцінної симуляції перехрестя і трафіку (рис. 3.6). Дані з OpenStreetMap не містять важливих деталей, таких як точні конфігурації перехресть, кількість світлофорів, точна ширина дороги та кути поворотів, що необхідні для точної симуляції складних дорожніх ситуацій. Тому для більш точного відтворення реальних умов руху було вирішено звернутися до спеціалізованих симуляторів, які пропонують більш детальну інформацію і можливості для моделювання трафіку.

```
</way>
<way id="159582307" visible="true" version="51"
  <nd ref="1358025007"/>
  <nd ref="3683006651"/>
  <nd ref="1716937315"/>
  <nd ref="10821417596"/>
  <nd ref="8219594107"/>
  <nd ref="7438424582"/>
  <nd ref="9524417651"/>
  <nd ref="10798859816"/>
  <nd ref="9524417648"/>
  <nd ref="1716937208"/>
  <nd ref="7464298925"/>
  <nd ref="4945293933"/>
  <tag k="highway" v="residential"/>
  <tag k="lanes" v="2"/>
  <tag k="maxwidth" v="2"/>
  <tag k="name" v="вул. Саксаганського"/>
  <tag k="name:en" v="Saksahanskoho Street"/>
  <tag k="surface" v="asphalt"/>
</way>
```

Рисунок 3.6 – Дані вулиці (формат OpenStreetMap)

Джерело: розроблено автором

Під час пошуку було обрано симулятор SUMO (Simulation of Urban Mobility), який забезпечує високу реалістичність моделювання трафіку, особливо на макроскопічному рівні, зокрема на перехрестях. SUMO підтримує створення

складних сценаріїв, імпорт карт, а також гнучке налаштування світлофорів [35] (рис. 3.7).

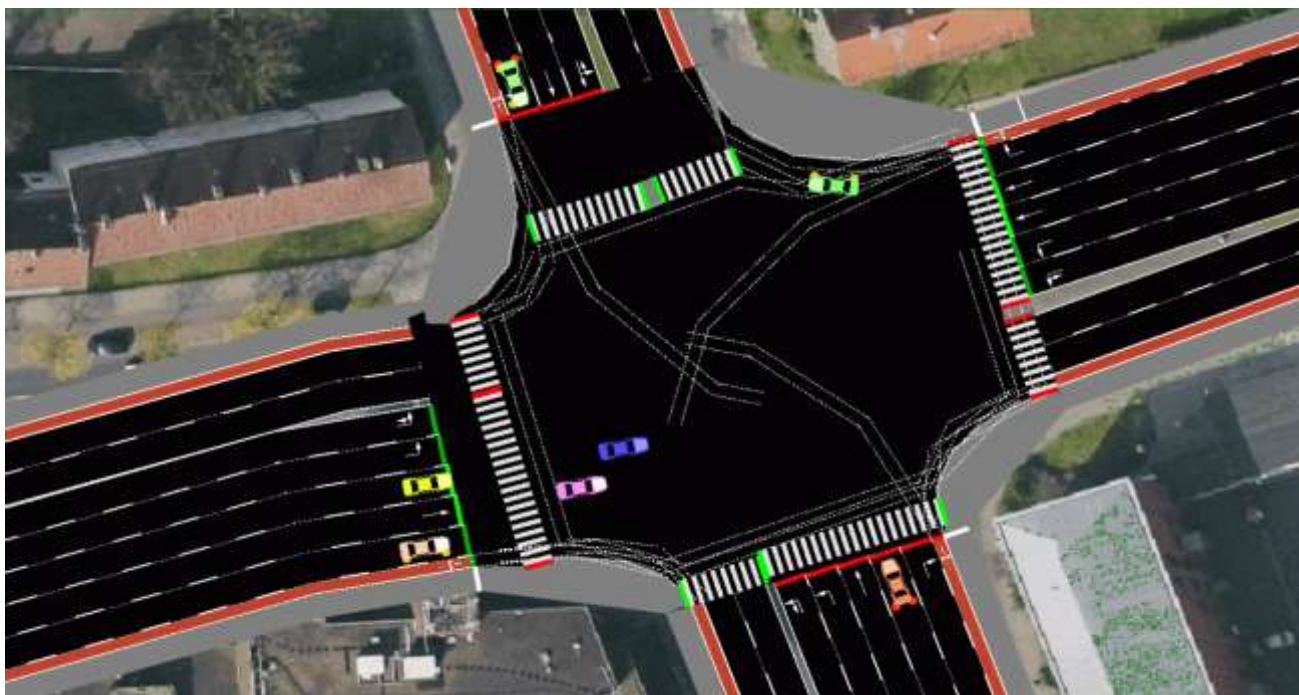


Рисунок 3.7 – Приклад деталізації перехрестя в SUMO

Джерело: [35]

На жаль, точних даних для реалістичного моделювання перехресть у відкритому доступі немає. Тому для навчання моделей будуть використовуватися стандартні конфігурації перехрестів з прикладів, наданих у SUMO. Хоча ці приклади мають достатню деталізацію для симуляції, для розробки реальної системи в майбутньому буде необхідний доступ до більш детальних даних для кожного конкретного перехрестя. Це дозволить точніше відтворювати реальні умови руху та враховувати індивідуальні особливості кожного перехрестя.

3.3 Визначення параметрів моделі

Для реалізації системи автономного управління дорожнім рухом на перехрестях було використано бібліотеку SUMO-RL, яка інтегрує симулятор дорожнього руху SUMO з методами навчання з підкріпленням [36]. Бібліотека забезпечує сумісність з Gymnasium, PettingZoo та популярними бібліотеками для навчання з підкріпленням, такими як Stable Baselines3 та RLlib, що дозволяє легко інтегрувати її в різні дослідницькі та прикладні проєкти.

Observation (спостереження). Стандартне спостереження для кожного агента (світлофора) є вектором, що включає:

- **phase_one_hot**: one-hot кодування поточної зеленої фази світлофора.
- **min_green**: бінарна ознака, що вказує, чи пройшов мінімальний зелений час.
- **lane_i_density**: щільність транспорту на входній смузі i (кількість транспортних засобів, поділена на місткість смуги).
- **lane_i_queue**: довжина черги на входній смузі i (кількість транспортних засобів зі швидкістю менше 0.1 м/с, поділена на місткість смуги).

Нижче представлено реалізацію функції, що підготовлює вектор з даними спостереження.

```
class DefaultObservationFunction(ObservationFunction):
    """Default observation function for traffic signals."""

    def __call__(self) -> np.ndarray:
        phase_id = [1 if self.ts.green_phase == i else 0 for i in
range(self.ts.num_green_phases)]
        min_green = [0 if self.ts.time_since_last_phase_change <
self.ts.min_green + self.ts.yellow_time else 1]
        density = self.ts.get_lanes_density()
        queue = self.ts.get_lanes_queue()
        observation = np.array(phase_id + min_green + density + queue,
dtype=np.float32)
        return observation
```

Джерело: [36]

Це спостереження можна змінити, реалізувавши власний клас, що успадковує **ObservationFunction**, та передавши його до конструктора середовища.

Action (дія). Простір дій є дискретним. Кожні **delta_time** секунд агент обирає наступну конфігурацію зеленої фази світлофора. Наприклад, для двостороннього перехрестя існує 4 дискретні дії, що відповідають різним конфігураціям зеленої фази. При зміні фази перед новою зеленою фазою завжди вставляється жовта фаза тривалістю **yellow_time** секунд.

Reward (винагорода). Стандартна функція винагороди базується на зміні сумарної затримки транспортних засобів.

У класі **TrafficSignal** реалізовано кілька вбудованих функцій винагороди, кожна з яких відображає різні аспекти ефективності управління трафіком. Наприклад, функція **diff-waiting-time** стимулює зменшення сумарного часу очікування транспортних засобів, порівнюючи поточне значення з попереднім. Якщо затримка зменшилась, агент отримує позитивну винагороду, що заохочує до зниження заторів.

Інші варіанти включають:

- **average-speed** – винагорода пропорційна середній швидкості транспортного потоку в зоні дії світлофора. Це стимулює підтримання руху без зупинок.
- **queue** – негативна винагорода за довгі черги, стимулює зменшення кількості зупинених авто.
- **pressure** – базується на різниці між кількістю транспортних засобів, що залишають перехрестя, і тих, що наближаються. Це дозволяє оптимізувати потік, уникаючи перевантаження.

Також є можливість реалізувати власну функцію винагороди та зареєструвати її через **TrafficSignal.register_reward_fn()**, що забезпечує гнучкість при моделюванні різних сценаріїв управління дорожнім рухом.

SUMO-RL підтримує як одноагентні, так і мультиагентні середовища, дозволяючи моделювати складні сценарії з кількома перехрестями. Бібліотека також надає приклади використання різних алгоритмів навчання з підкріпленням,

таких як Q-learning, SARSA, PPO та DQN, що полегшує експерименти та порівняння методів.

3.4 Тренування та порівняння моделей

Навчання алгоритмів проводилося на типовій конфігурації Х-подібного перехрестя з двостороннім рухом (рис. 3.8).

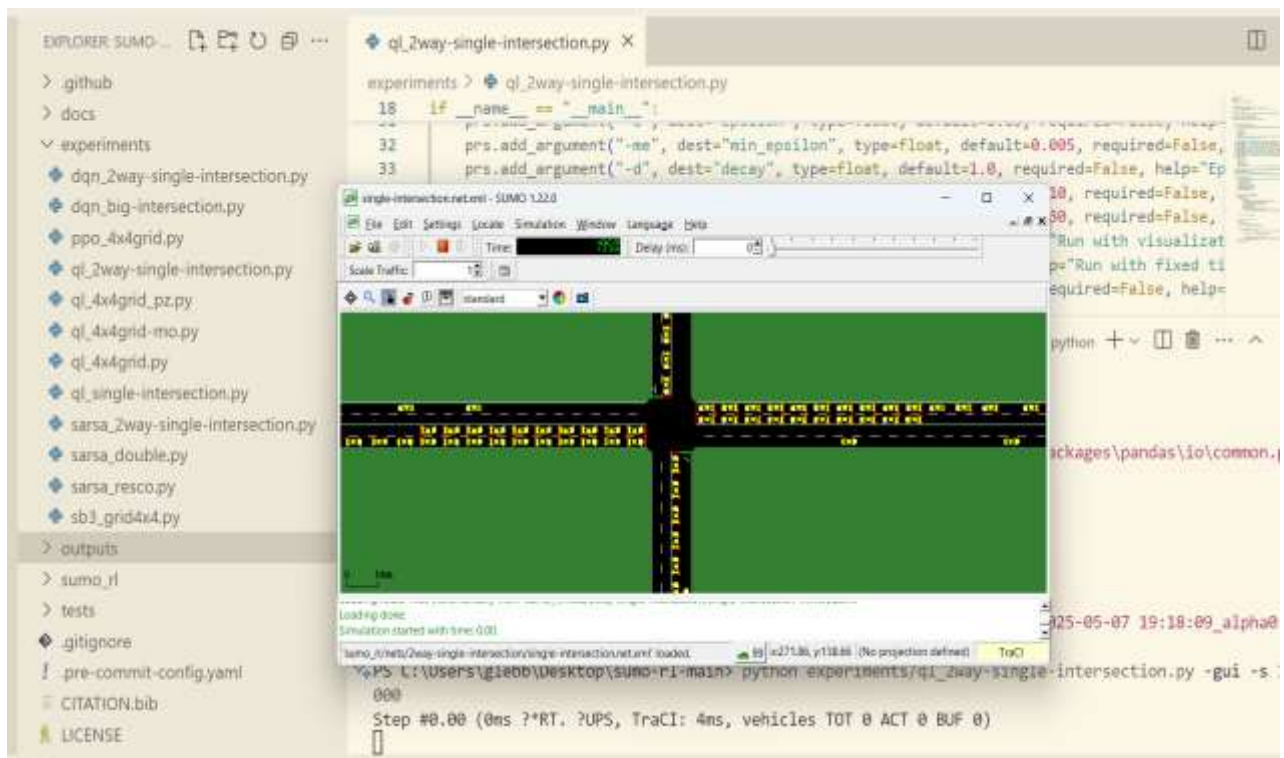
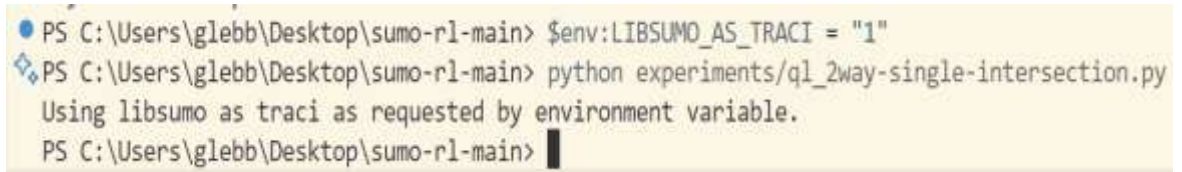


Рисунок 3.8 – Процес тренування використовуючи графічний інтерфейс SUMO

Джерело: розроблено автором

Тренування можна проводити використовуючи графічний інтерфейс але такий спосіб використовується лише для перевірки роботи моделей. Тренування моделей проводилось із використанням libsumo – ефективного інтерфейсу до симулятора SUMO, який дозволяє взаємодіяти з симуляцією без запуску графічного

інтерфейсу [37]. Це суттєво пришвидшує процес навчання, оскільки зменшує навантаження на систему (рис. 3.9).



```
PS C:\Users\glebb\Desktop\sumo-rl-main> $env:LIBSUMO_AS_TRACI = "1"
PS C:\Users\glebb\Desktop\sumo-rl-main> python experiments/ql_2way-single-intersection.py
Using libsumo as traci as requested by environment variable.
PS C:\Users\glebb\Desktop\sumo-rl-main> █
```

Рисунок 3.9 – Тренування моделей використовуючи libsumo

Джерело: розроблено автором

Libsumo працює як бібліотека на рівні процесу, на відміну від TraCI, який використовує міжпроцесну комунікацію, що робить libsumo швидшим та зручнішим для паралельних запусків. Такий підхід дозволив ефективно проводити багаточисельні епохи навчання, мінімізуючи час симуляції та забезпечуючи кращу інтеграцію з reinforcement learning-середовищем.

Нижче наведено налаштування середовища для тренування моделі використовуючи алгоритм Q-learning:

```
env = SumoEnvironment(
    net_file="sumo_rl/nets/2way-single-intersection/single-
intersection.net.xml",
    route_file=args.route,
    out_csv_name=out_csv,
    use_gui=args.gui,
    num_seconds=args.seconds,
    min_green=args.min_green,
    max_green=args.max_green,
    sumo_warnings=False,
)

for run in range(1, args.runs + 1):
    initial_states = env.reset()
    ql_agents = {
        ts: QLAgent(
            starting_state=env.encode(initial_states[ts], ts),
            state_space=env.observation_space,
            action_space=env.action_space,
            alpha=args.alpha,
            gamma=args.gamma,
            exploration_strategy=EpsilonGreedy(
```

```

        initial_epsilon=args.epsilon,
min_epsilon=args.min_epsilon, decay=args.decay
    ),
    )
    for ts in env.ts_ids
}

done = {"__all__": False}
infos = []
if args.fixed:
    while not done["__all__"]:
        _, _, done, _ = env.step({})
else:
    while not done["__all__"]:
        actions = {ts: ql_agents[ts].act() for ts in
ql_agents.keys()}

        s, r, done, _ = env.step(action=actions)

        for agent_id in ql_agents.keys():
            ql_agents[agent_id].learn(next_state=env.encode(s[agent
_id], agent_id), reward=r[agent_id])
        env.save_csv(out_csv, run)
        env.close()

```

Подібним чином було налаштовано тренування моделі використовуючи алгоритм SARSA:

```

env = SumoEnvironment(
    net_file="sumo_rl/nets/2way-single-intersection/single-
intersection.net.xml",
    single_agent=True,
    route_file=args.route,
    out_csv_name=out_csv,
    use_gui=args.gui,
    num_seconds=args.seconds,
    min_green=args.min_green,
    max_green=args.max_green,
)

for run in range(1, args.runs + 1):
    obs, info = env.reset()
    agent = TrueOnlineSarsaLambda(
        env.observation_space,
        env.action_space,
        alpha=args.alpha,
        gamma=args.gamma,
        epsilon=args.epsilon,

```

```

        fourier_order=7,
        lamb=0.95,
    )

    terminated, truncated = False, False
    if args.fixed:
        while not (terminated or truncated):
            _, _, terminated, truncated, _ = env.step({})
    else:
        while not (terminated or truncated):
            action = agent.act(obs)

            next_obs, r, terminated, truncated, info =
env.step(action=action)

            agent.learn(state=obs, action=action, reward=r,
next_state=next_obs, done=terminated)

            obs = next_obs

    env.save_csv(out_csv, run)

```

Для налаштування середовища тренування використовуючи алгоритм DQN були використано бібліотеку **stable-baselines3** [38]. Це високорівнева бібліотека для реалізації алгоритмів навчання з підкріпленням, що забезпечує зручне використання перевірених методів, включаючи DQN. У процесі тренування використовувались доступні в SB3 функції для збереження моделей, логування метрик, а також конфігурації частоти оновлення політики, буфера досвіду та ϵ -жадібної стратегії. Бібліотека дозволяє задавати гіперпараметри моделі, такі як розмір нейронної мережі, швидкість навчання, розмір батчу та довжина replay buffer, що забезпечує гнучкість при адаптації до особливостей трафіку на перехрестях.

```

from stable_baselines3.dqn.dqn import DQN

...

env = SumoEnvironment(
    net_file="sumo_rl/nets/2way-single-intersection/single-
intersection.net.xml",
    route_file="sumo_rl/nets/2way-single-intersection/single-
intersection-vhvh.rou.xml",

```

```

        out_csv_name="outputs/2way-single-intersection/dqn",
        single_agent=True,
        use_gui=False,
        num_seconds=100000,
    )

    model = DQN(
        env=env,
        policy="MlpPolicy",
        learning_rate=0.001,
        learning_starts=0,
        train_freq=1,
        target_update_interval=500,
        exploration_initial_eps=0.05,
        exploration_final_eps=0.01,
        verbose=1,
    )
    model.learn(total_timesteps=100000)

```

Результати навчання представлені у вигляді графіків, які демонструють зміну цільової метрики (загального часу очікування) залежно від епізодів тренування для кожного з використаних алгоритмів. Це дозволяє оцінити ефективність навчання та швидкість збіжності моделей. З графіків видно, як різні підходи до управління світлофорами впливають на зменшення заторів у процесі моделювання (рис. 3.10 – 3.16).

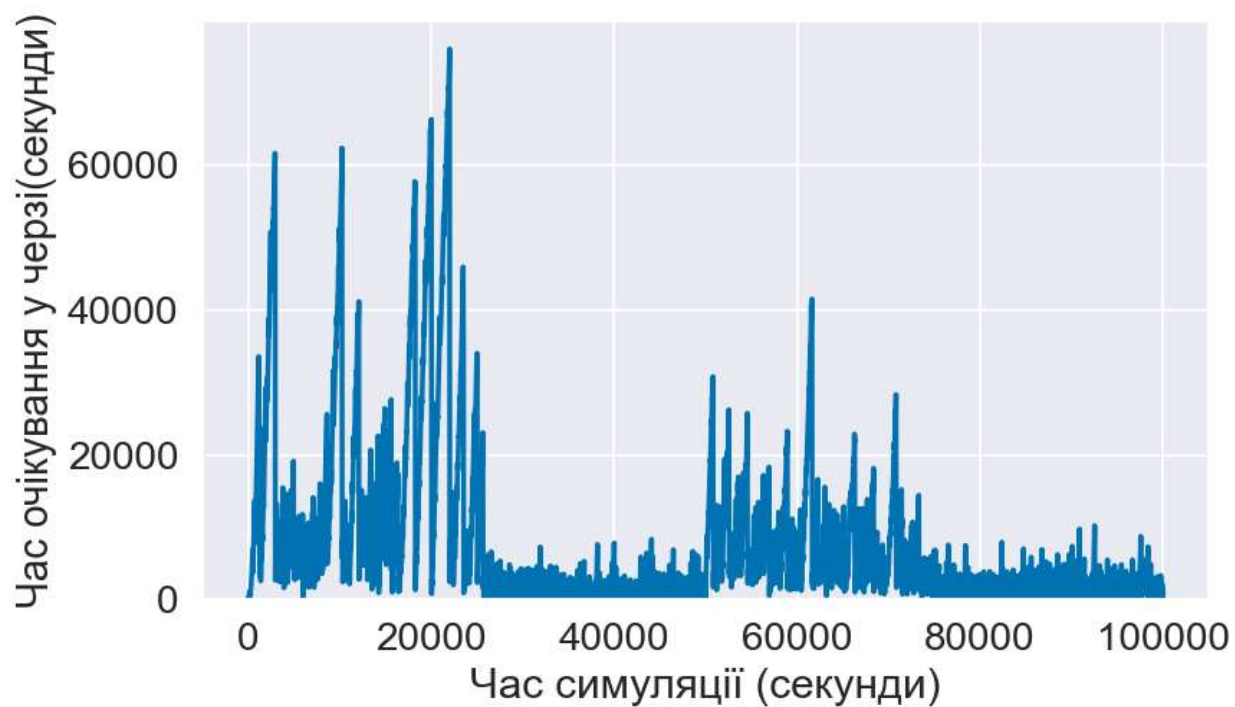


Рисунок 3.10 – Результати тренування моделі з Q-learning

Джерело: розроблено автором

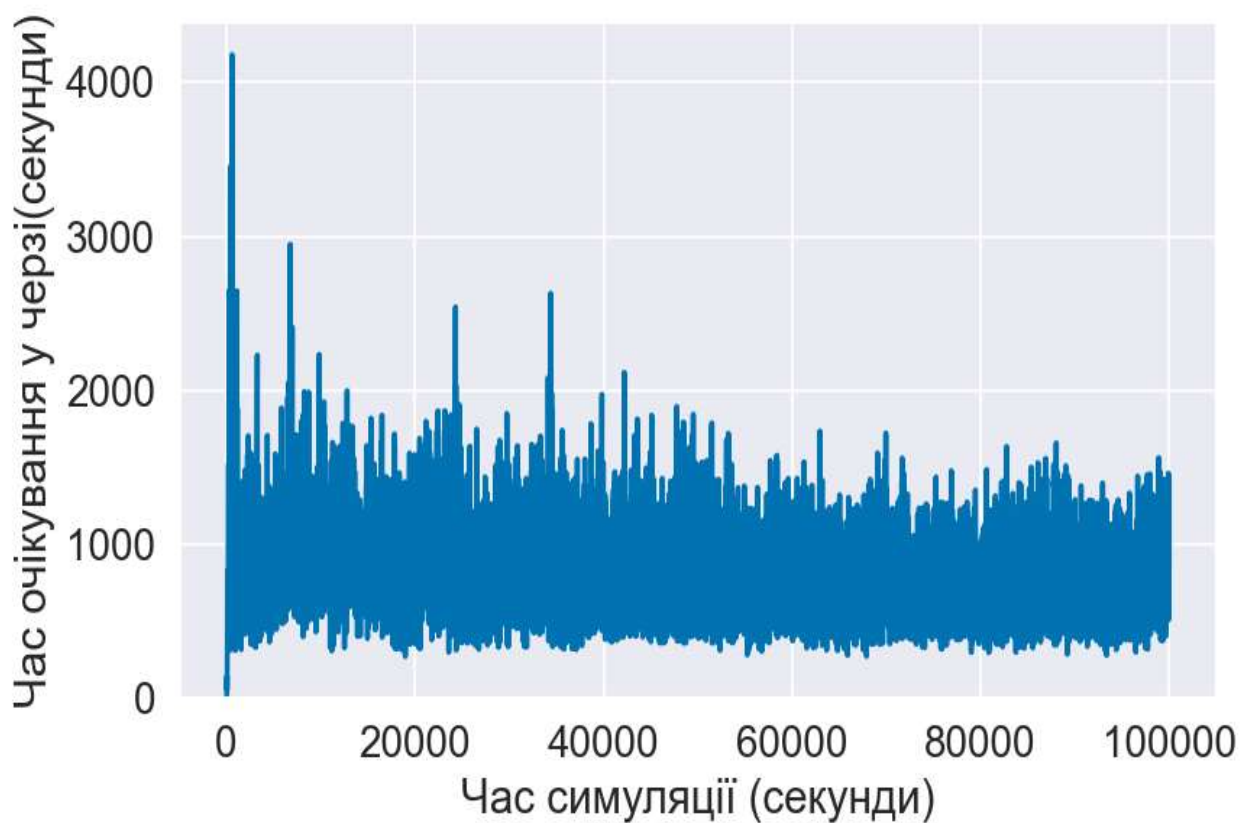


Рисунок 3.11 – Результати тренування моделі з SARSA

Джерело: розроблено автором

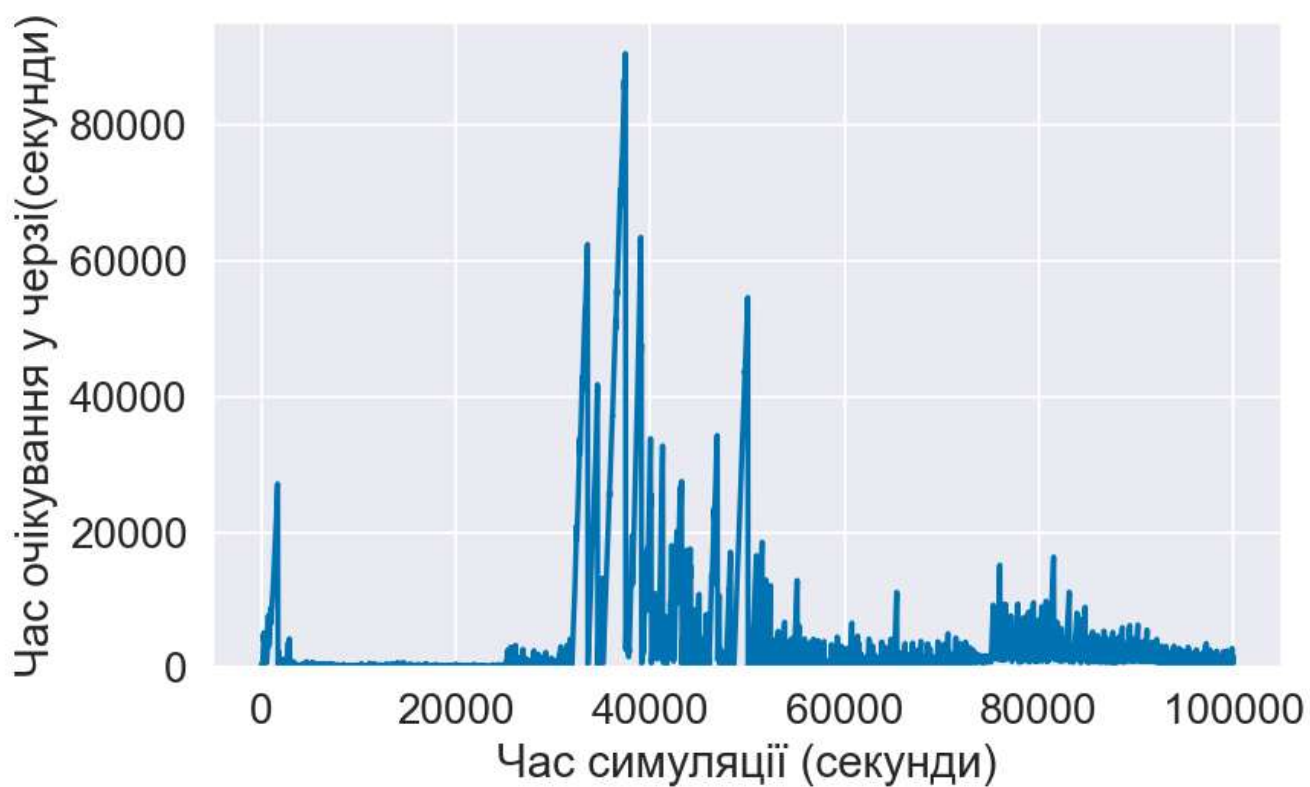


Рисунок 3.12 – Результати тренування моделі з DQN (епізод 1)

Джерело: розроблено автором

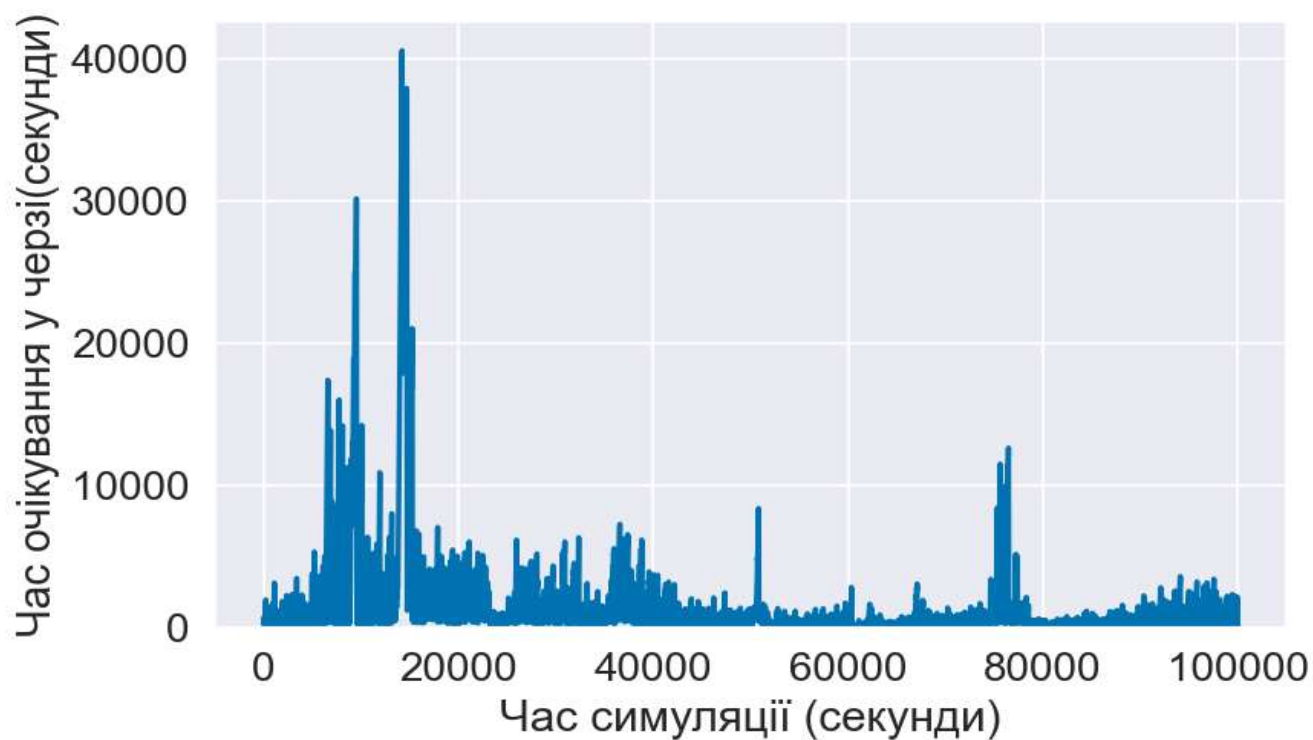


Рисунок 3.13 – Результати тренування моделі з DQN (епізод 2)

Джерело: розроблено автором

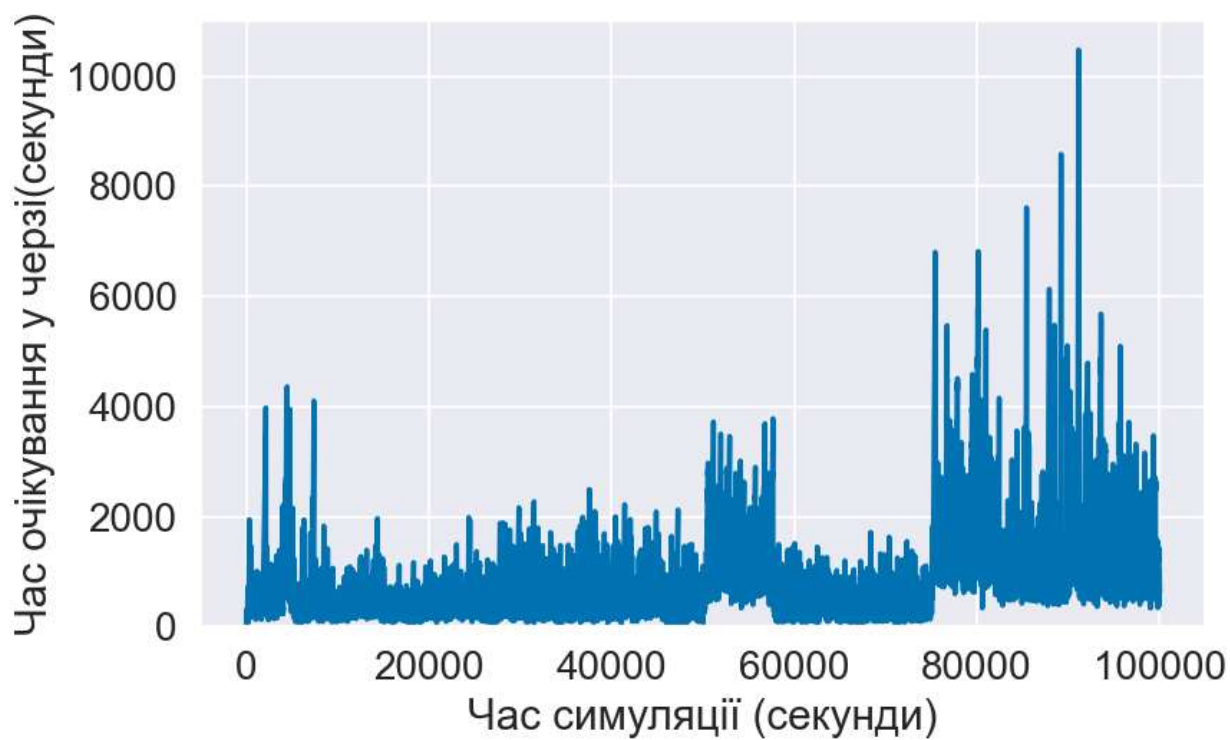


Рисунок 3.14 – Результати тренування моделі з DQN (епізод 3)

Джерело: розроблено автором

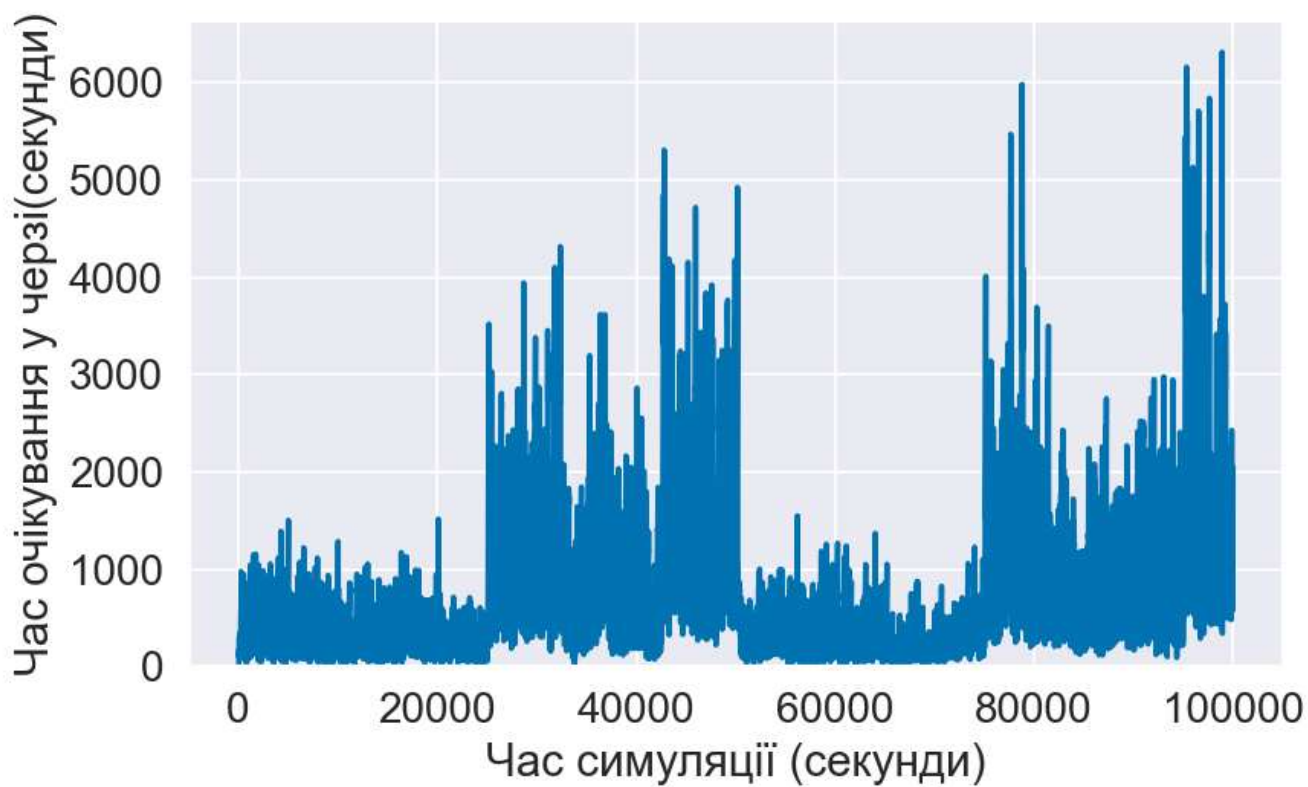


Рисунок 3.15 – Результати тренування моделі з DQN (епізод 4)

Джерело: розроблено автором

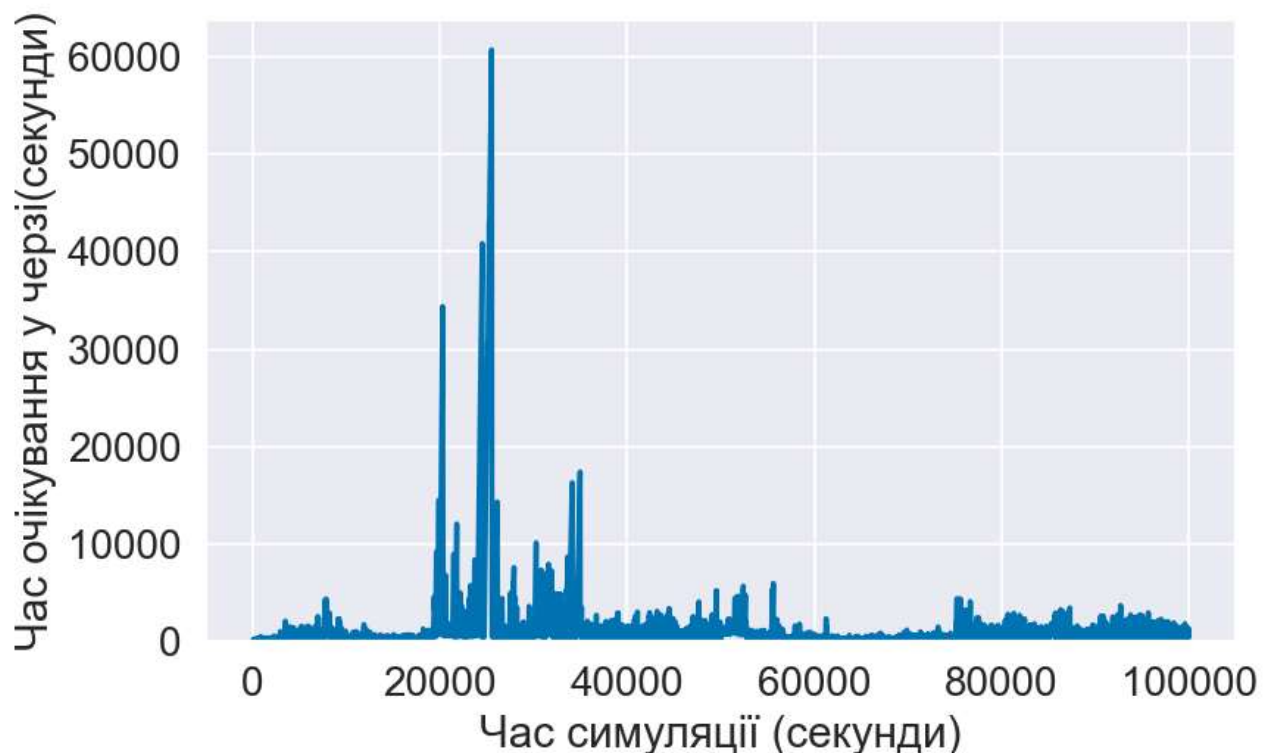


Рисунок 3.16 – Результати тренування моделі з DQN (епізод 5)

Джерело: розроблено автором

Згідно результатів тренування, візуалізованими на рисунках 3.10 – 3.16, алгоритми Q-learning та SARSA продемонстрували різні патерни навчання у задачі оптимізації трафіку. Модель Q-learning (рис. 3.10) на початкових етапах симуляції мала значні коливання часу очікування з високими піками, що свідчить про фазу активного дослідження. Однак, згодом спостерігалася швидка конвергенція до значно нижчих та стабільніших значень загального часу очікування. Натомість, модель SARSA (рис. 3.11) показала менші початкові піки, але її продуктивність характеризувалася постійними коливаннями часу очікування протягом усієї симуляції, тримаючись на відносно стабільному середньому рівні.

Результати тренування моделі DQN протягом п'яти епізодів (рисунки 3.12 – 3.16) чітко ілюструють процес навчання. Ранні епізоди (рис. 3.12, 3.13) відзначені високою волатильністю та наявністю значних піків часу очікування. З кожним наступним епізодом (рис. 3.14, 3.15) амплітуда та частота цих піків поступово зменшувалися. У пізніх епізодах, зокрема в п'ятому (рис. 3.16), модель DQN досягла найнижчих та найбільш стабільних значень загального часу очікування серед усіх

протестованих підходів, що підтверджує ефективність глибинного навчання з підкріпленням для даної задачі після достатньої кількості ітерацій тренування. Ці графіки наочно ілюструють, як різні алгоритми навчання з підкріпленням, а також тривалість тренування (у випадку DQN), впливають на здатність системи управління світлофорами оптимізувати потік трафіку та зменшувати затори, виміряні за загальним часом очікування транспортних засобів у черзі.

Згідно з результатами проведеного тренування представлених на графіках у попередніх розділах було відзначено, що алгоритм Q-learning, попри свою відносну простоту порівняно з глибинними методами, продемонстрував достатньо хороші результати у зменшенні загального часу очікування після етапу початкового дослідження середовища. Це робить його придатним кандидатом для подальших етапів роботи, включаючи візуалізацію та перевірку працездатності в умовах, наближених до реальних.

3.5 Візуалізація та перевірка результатів

Для візуального підтвердження працездатності розробленої системи управління світлофорами була реалізована спрощена симуляційна візуалізація роботи світлофора та реакції на його дії з боку транспортних засобів. Цей інструмент не тільки надає наочне уявлення про поведінку системи, але й дозволяє відтворювати її роботу без необхідності залучення повноцінного середовища машинного навчання, що є особливо важливим для потенційного розгортання на малопотужному обладнанні.

В умовах відсутності доступу до реального обладнання, створена симуляція прагне максимально наблизитись до реальних умов трафіку.

Для цього було використано 3D-моделювання за допомогою бібліотеки three.js [39], що дозволяє створювати інтерактивні та динамічні візуалізації. Three.js надає потужні інструменти для побудови тривимірних сцен (рис. 3.17), що

дозволяють відтворювати рух транспорту та взаємодію з світлофорами в реальному часі.

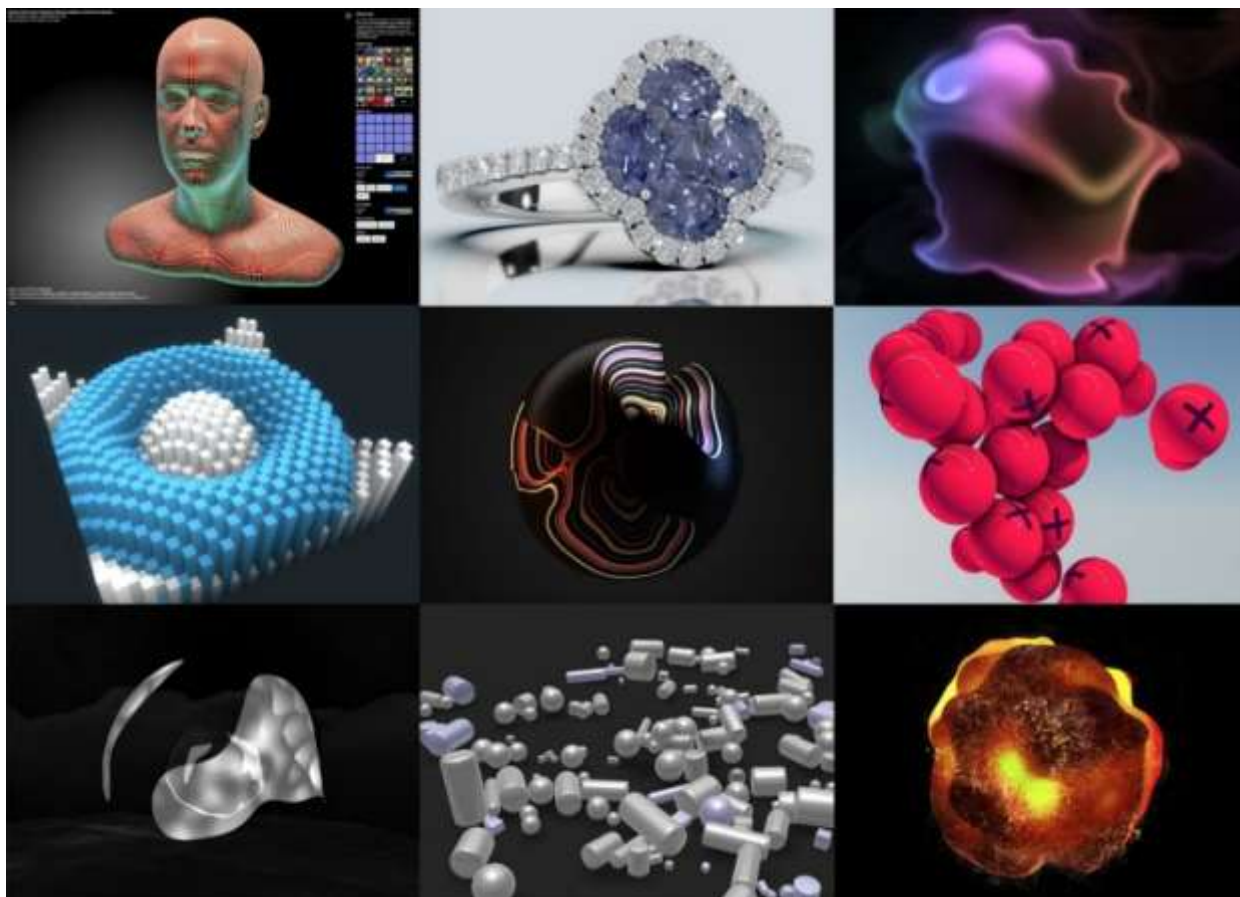


Рисунок 3.17 – Можливості бібліотеки three.js

Джерело: [40]

Для демонстрації в симуляції було використано модель, навчення якої здійснювалося за алгоритмом Q-learning через її простоту та достатньо хороші результати тренування.

На рисунку 3.17 представлена загальна візуалізація симуляційного середовища в процесі роботи. Графічний інтерфейс відображає перехрестя, рух транспортних засобів по смугах, а також елементи керування та інформаційні панелі, включаючи показники часу симуляції та прогресу.

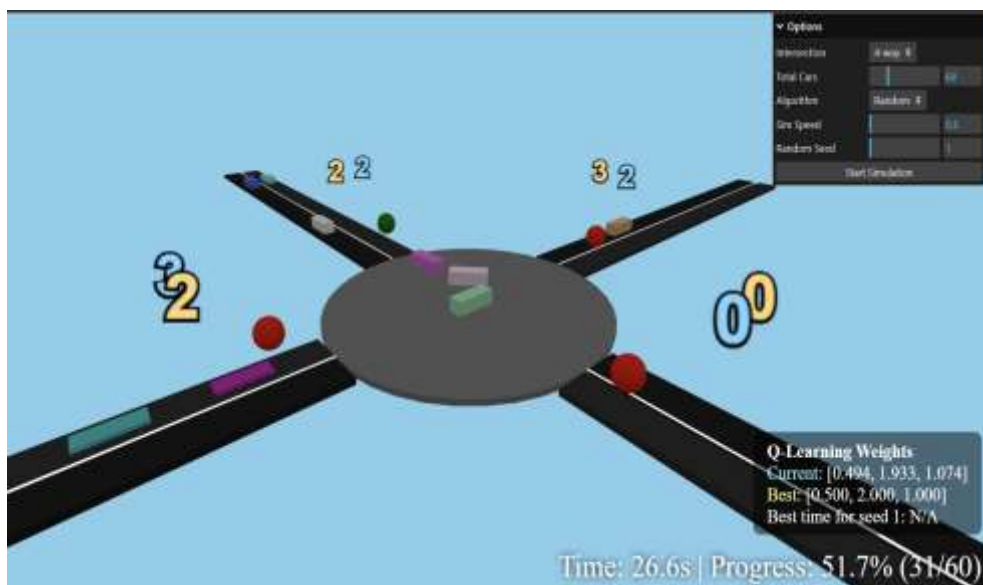


Рисунок 3.18 – Система в роботі

Джерело: розроблено автором

Для порівняльного аналізу ефективності прийнятих рішень було проведено симуляцію в режимі випадкового вибору фази світлофора. Результат такої симуляції, що виступає як базовий сценарій, показано на рисунку 3.18. В цьому режимі фази перемикаються без урахування реального трафіку, що, як правило, призводить до неефективного розподілу часу проїзду та утворення заторів.

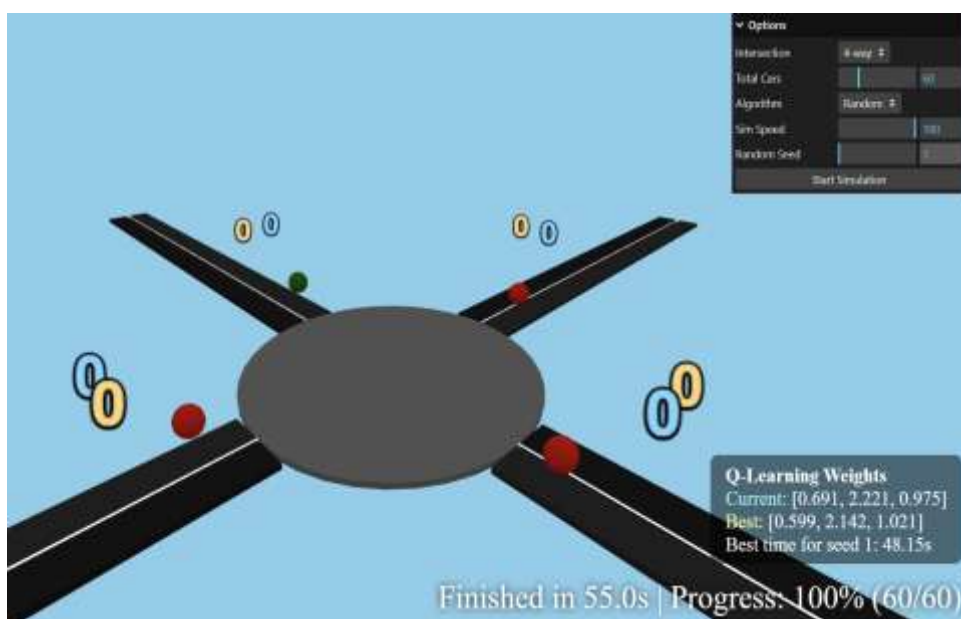


Рисунок 3.19 – Результат симуляції в режимі випадкового вибору фази

Джерело: розроблено автором

На відміну від випадкового режиму, симуляція з використанням навченої моделі Q-learning демонструє адаптивне управління світлофором. Рисунок 3.19 ілюструє стан перехрестя під керуванням агента Q-learning. Можна спостерігати, як система реагує на наявність транспортних засобів на різних підходах до перехрестя, оптимізуючи послідовність та тривалість зелених фаз з метою мінімізації часу очікування. Порівняння з рисунком 3.18 візуально підтверджує переваги інтелектуального управління.

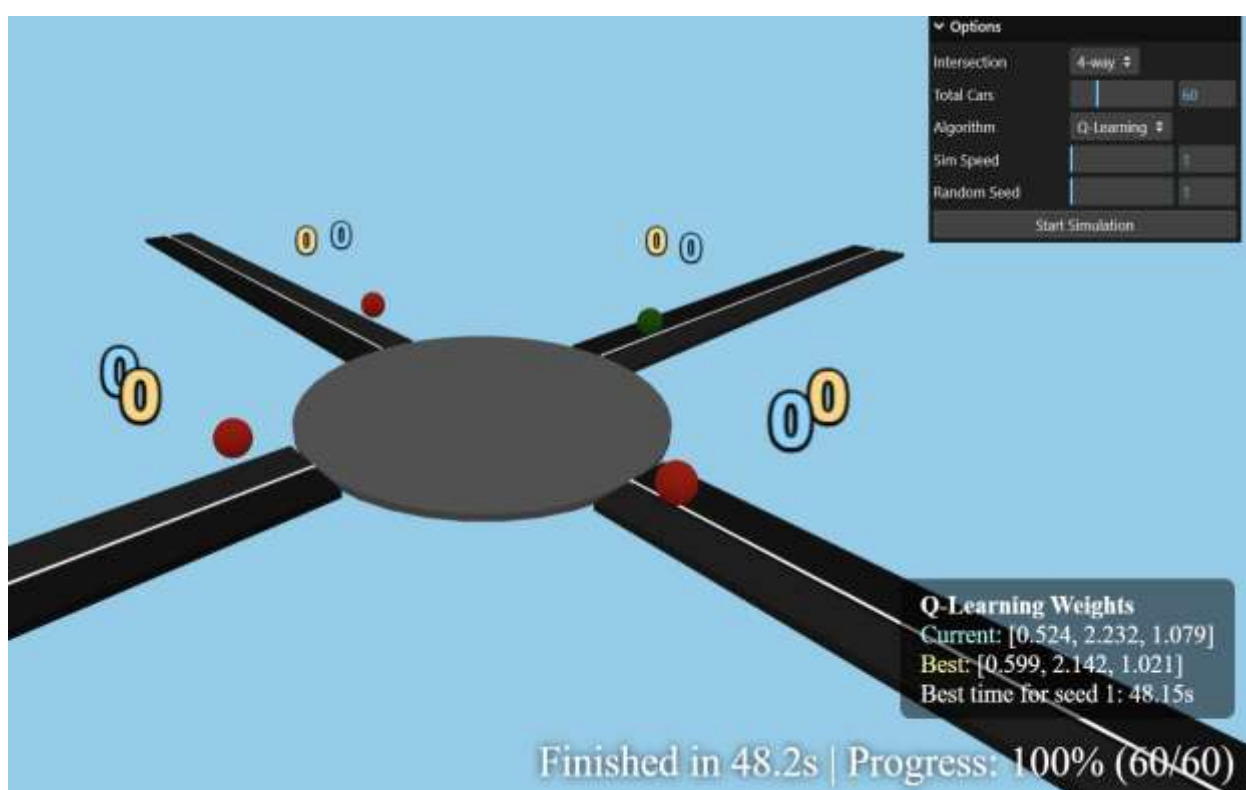


Рисунок 3.20 – Результат симуляції з використанням Q-learning

Джерело: розроблено автором

Створене симуляційне середовище також передбачає можливість моделювання перехресть різної конфігурації. На рисунку 3.20 показано приклад візуалізації Т-подібного перехрестя. Це демонструє гнучкість розробленої системи та її потенційну адаптивність до різних топологій міської дорожньої мережі, що є важливим аспектом для практичного застосування.



Рисунок 3.21 – Конфігурація з Т-подібним перехрестям

Джерело: розроблено автором

Реалізована візуалізація та можливість тестування навчених моделей у симуляційному середовищі є важливим етапом для перевірки коректності роботи алгоритмів та наочної демонстрації досягнутих результатів перед переходом до реальних сценаріїв використання.

Висновки до Розділу 3

У даному розділі було детально описано процес розробки та оцінки системи автономного управління світлофорами на основі методів навчання з підкріпленням. Представлено архітектуру симуляційного середовища, побудованого з використанням SUMO та GYM, що дозволило створити реалістичну модель перехрестя та процесу руху транспорту. Досліджено застосування трьох ключових алгоритмів навчання з підкріпленням: Q-learning, SARSA та DQN.

Проведене тренування продемонструвало, що кожен з вибраних алгоритмів здатен до навчання та оптимізації управління світлофором з метою мінімізації загального часу очікування транспортних засобів у черзі. Аналіз графіків тренувального процесу показав, що хоча простіші алгоритми, такі як Q-learning, швидко досягають стабільного стану зі зниженим часом очікування, більш складний алгоритм DQN, особливо після кількох епізодів тренування, демонструє кращу продуктивність у зменшенні пікових навантажень та забезпеченні стабільно низького часу затримок. Ці результати підтверджують ефективність обраного підходу до вирішення задачі оптимізації трафіку.

Реалізована спрощена візуалізація симуляційного середовища стала важливим інструментом для наочної демонстрації роботи навченої системи та перевірки її поведінки в умовах, максимально наближених до реальних. Використання навченої моделі Q-learning у візуалізації дозволило підтвердити практичну придатність отриманих в результаті навчання політик управління.

Загалом, результати, отримані в симуляційному середовищі, підтверджують працездатність та потенційну ефективність запропонованої системи автономного управління світлофорами на основі навчання з підкріпленням. Розроблений підхід формує міцну основу для подальших досліджень та може бути масштабований для впровадження у реальних міських умовах, за умов наявності необхідної апаратної інфраструктури, детальних даних про топологію дорожньої мережі та актуальної інформації про транспортні потоки.

ВИСНОВКИ

У межах виконання кваліфікаційної магістерської роботи було успішно вирішено задачу розроблення системи автономного управління дорожнім рухом на перехрестях з використанням алгоритмів штучного інтелекту.

Проведено всебічний аналіз предметної області, існуючих підходів та технологій, що дозволило обґрунтувати вибір методів машинного навчання, зокрема підкріплювального навчання, як найбільш перспективних для адаптивного керування трафіком у динамічних міських умовах.

Досліджено архітектуру та компоненти такої системи, враховуючи можливості інтеграції з типовим існуючим обладнанням світлофорних об'єктів.

Ключовим етапом роботи стала розробка та реалізація симуляційного середовища на базі SUMO, що забезпечило реалістичні умови для тренування та тестування алгоритмів. У цьому середовищі було здійснено тренування моделей з використанням алгоритмів Q-learning, SARSA та DQN.

Отримані результати підтвердили здатність цих алгоритмів до навчання та оптимізації управління світлофорами з метою мінімізації загального часу очікування транспортних засобів. Зокрема, показано, що алгоритм DQN демонструє найвищу ефективність у зменшенні заторів після достатньої кількості ітерацій тренування.

Додатково розроблена спрощена візуалізація процесу симуляції дозволила наочно продемонструвати працездатність системи та поведінку навченого агента в імітованих умовах.

Результати проведеного дослідження свідчать про практичну значимість розробленої системи та підтверджують доцільність її подальшого впровадження. Запропонований підхід може бути використаний для створення реальних систем автономного управління світлофорами на перехрестях українських міст, що сприятиме підвищенню пропускної спроможності дорожньої мережі та безпеки дорожнього руху.

Для успішної реалізації на практиці необхідне забезпечення доступу до апаратної частини світлофорних контролерів, отримання детальних конфігурацій конкретних перехресть та організація збору якісних даних про транспортні потоки для адаптації та донавчання моделей в реальних умовах експлуатації. Врахування специфіки нестабільного трафіку та інтеграція алгоритмів в ресурсообмежене обладнання є важливими напрямками для подальших досліджень та вдосконалення системи.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. SCOOT. Traffic signal optimisation and coordination for connected networks [Електронний ресурс] – Режим доступу до ресурсу: <https://trlsoftware.com/software/intelligent-signal-control/scoot/>
2. SCATS. An intelligent, adaptive traffic control system [Електронний ресурс] – Режим доступу до ресурсу: <https://www.scats.nsw.gov.au/home>
3. Автоматизована система керування дорожнім рухом. Програмне забезпечення АСКДР SEA TCS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sea.com.ua/ua/svetofornaya-produkciya/kompleksnaa-sistema-upravlenia-doroznym-dvizeniem/>
4. програмне забезпечення АСКДР SEA TCS [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sea.com.ua/ua/svetofornaya-produkciya/programmnoe-obespecenie/>
5. Дорожні контролери [Електронний ресурс] – Режим доступу до ресурсу: <https://www.sea.com.ua/ua/svetofornaya-produkciya/doroznye-kontrollery/>
6. Smart traffic management of vehicles using faster R-CNN based deep learning method [Електронний ресурс] – Режим доступу до ресурсу: <https://www.nature.com/articles/s41598-024-60596-4>
7. What is R-CNN? [Електронний ресурс] – Режим доступу до ресурсу: <https://blog.roboflow.com/what-is-r-cnn/>
8. Eclipse SUMO (Simulation of Urban MObility) [Електронний ресурс] – Режим доступу до ресурсу: <https://eclipse.dev/sumo/>
9. Analyzing the robustness of adaptive traffic control system using reinforcement learning for urban traffic flow [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/390893204_ANALYZING_THE_ROBUSTNESS_OF_ADAPTIVE_TRAFFIC_CONTROL_SYSTEM_USING_REINFORCEMENT_LEARNING_FOR_URBAN_TRAFFIC_FLOW
10. Artificial Intelligence in Intelligent Traffic Signal Control [Електронний ресурс] – Режим доступу до ресурсу: <https://www.researchgate.net/publication/389019464>
11. Smart-VDCTC: Smart vehicular detection classification and traffic control for Indian urban traffic [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/389876782_Smart-VDCTC_Smart_vehicular_detection_classification_and_traffic_control_for_Indian_urban_traffic
12. Autonomous Driving Strategies at Intersections: Scenarios, State-of-the-Art, and Future Outlooks [Електронний ресурс] – Режим доступу до ресурсу: https://www.researchgate.net/publication/352523373_Autonomous_Driving_Strategies_at_Intersections_Scenarios_State-of-the-Art_and_Future_Outlooks

13. movement of cars and pedestrians crossing the streets of Buenos Aires. Argentina. South America. [Електронний ресурс] – Режим доступу до ресурсу: <https://www.alamy.com/stock-photo-movement-of-cars-and-pedestrians-crossing-the-streets-of-buenos-aires-93552855.html>
14. Best-of Machine Learning with Python [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/ml-tooling/best-of-ml-python>
15. Data Science Tools: The Ultimate Guide. [Електронний ресурс] – Режим доступу до ресурсу: <https://medium.com/@fosberg1addai/data-science-tools-the-ultimate-guide-cf01b73fe3ee>
16. TraCI - Traffic Control Interface [Електронний ресурс] – Режим доступу до ресурсу: <https://pypi.org/project/traci/>
17. JupyterLite [Електронний ресурс] – Режим доступу до ресурсу: <https://jupyter.org/try-jupyter/lab/>
18. Eclipse SUMO - Simulation of Urban MObility [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/eclipse-sumo/sumo>
19. D3 by Observable | The JavaScript library for bespoke data visualization [Електронний ресурс] – Режим доступу до ресурсу: <https://d3js.org/>
20. VS Code | Your code editor. Redefined with AI. [Електронний ресурс] – Режим доступу до ресурсу: <https://code.visualstudio.com/>
21. GitHub Copilot | Your AI pair programmer [Електронний ресурс] – Режим доступу до ресурсу: <https://github.com/features/copilot>
22. Autonomous Driving Strategies at Intersections: Scenarios, State-of-the-Art, and Future Outlooks [Електронний ресурс] – Режим доступу до ресурсу: <https://www.catalyzex.com/paper/autonomous-driving-strategies-at>
23. Критерії застосування світлофорних об'єктів [Електронний ресурс] – Режим доступу до ресурсу: https://elib.lntu.edu.ua/sites/default/files/elib_upload/%D0%A1%D0%BC%D0%B0%D0%BB%D1%8C%20%D0%BF%D1%96%D0%B4%D1%80%D1%83%D1%87%D0%BD%D0%B8%D0%BA/page14.html
24. Traffic devices: historical aspects thereof [Електронний ресурс] – Режим доступу до ресурсу: <https://search.worldcat.org/title/278619>
25. Traffic Signal Timing Manual [Електронний ресурс] – Режим доступу до ресурсу: <https://ops.fhwa.dot.gov/publications/fhwahop08024/chapter5.htm#:~:text=Fully%20actuated%20control%20refers%20to,the%20course%20of%20the%20day.>
26. Reinforcement learning [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Reinforcement_learning
27. Кратко про алгоритм обучения Q-learning и как он реализуется в Python для новичков [Електронний ресурс] – Режим доступу до ресурсу: <https://habr.com/ru/companies/otus/articles/803041/>
28. Bellman equation [Електронний ресурс] – Режим доступу до ресурсу: https://en.wikipedia.org/wiki/Bellman_equation

29. SARSA. State–action–reward–state–action [Электронный ресурс] – Режим доступа до ресурсу: <https://en.wikipedia.org/wiki/State%E2%80%93action%E2%80%93reward%E2%80%93state%E2%80%93action>
30. Q-Learning vs. Deep Q-Learning vs. Deep Q-Network [Электронный ресурс] – Режим доступа до ресурсу: <https://www.baeldung.com/cs/q-learning-vs-deep-q-learning-vs-deep-q-network>
31. Train Your First Deep Q Learning based RL Agent: A Step-by-Step Guide [Электронный ресурс] – Режим доступа до ресурсу: <https://becominghuman.ai/train-your-first-rl-agent-a-step-by-step-guide-353bcd83722>
32. Playing Atari with Deep Reinforcement Learning [Электронный ресурс] – Режим доступа до ресурсу: <https://arxiv.org/pdf/1312.5602>
33. Export | OpenStreetMap [Электронный ресурс] – Режим доступа до ресурсу: <https://www.openstreetmap.org/export#map=15/50.35974/30.42043>
34. Google Maps [Электронный ресурс] – Режим доступа до ресурсу: <https://www.google.com/maps>
35. SUMO—a Traffic Simulator over the Cloud with Yonohub [Электронный ресурс] – Режим доступа до ресурсу: <https://ahmedhendawy.de/blog/2020/sumo/>
36. SUMO-RL. Reinforcement Learning environments for Traffic Signal Control with SUMO. Compatible with Gymnasium, PettingZoo, and popular RL libraries. [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/LucasAlegre/sumo-rl>
37. Libsumo - SUMO Documentation [Электронный ресурс] – Режим доступа до ресурсу: <https://sumo.dlr.de/docs/Libsumo.html>
38. stable-baselines3. PyTorch version of Stable Baselines, reliable implementations of reinforcement learning algorithms. [Электронный ресурс] – Режим доступа до ресурсу: <https://github.com/DLR-RM/stable-baselines3>
39. Three.js – JavaScript 3D Library. [Электронный ресурс] – Режим доступа до ресурсу: <https://threejs.org/>
40. Awesome Examples of Three.js [Электронный ресурс] – Режим доступа до ресурсу: <https://henryegloff.com/awesome-examples-of-three-js/>

ДОДАТКИ

Додаток А

Програмний код першого прототипу симулятора трафіку

```
import * as THREE from "three";
import { MapControls } from "three/addons/controls/MapControls.js";
import type { OSMJson, OSMNode } from "../lib/osm-to-json.mts";
import GUI from "lil-gui";

// converted using `../lib/osm-to-json.mts` from the following source:
https://www.openstreetmap.org/export#map=15/50.35974/30.42043
const map: OSMJson = await fetch("/map.json").then((r) => r.json());
const routeMap = buildRouteMap(map);

main();

function main() {
  const gui = new GUI({ title: "Options" });

  const canvas = document.querySelector("canvas")! as HTMLCanvasElement;

  const scene = new THREE.Scene();
  scene.background = new THREE.Color("blue");

  const renderer = new THREE.WebGLRenderer({ canvas, antialias: true });
  renderer.shadowMap.enabled = false;
  renderer.setPixelRatio(window.devicePixelRatio);
  renderer.setSize(window.innerWidth, window.innerHeight);
  renderer.setAnimationLoop(animate);

  const camera = new THREE.PerspectiveCamera(
    60,
    window.innerWidth / window.innerHeight,
    1,
    2000
  );
  camera.position.set(0, 200, -400);

  const controls = new MapControls(camera, renderer.domElement);
  controls.enableDamping = true;
  controls.dampingFactor = 0.1;
  controls.maxPolarAngle = Math.PI / 2;

  const light = new THREE.HemisphereLight("white", "white", 1);
  light.position.set(0, 500, 0);
  scene.add(light);
}
```

```

const floor = new THREE.Mesh(
  new THREE.PlaneGeometry(1200, 1200, 1, 1),
  new THREE.MeshPhongMaterial({
    color: "white",
    shininess: 0,
  })
);
floor.rotation.x = -Math.PI / 2;
floor.position.y = -0.1;
scene.add(floor);

const roads = new THREE.Group();
scene.add(roads);
// NOTE: no await here to load in background
addInBatchesStreaming(roads, loadRoads(map), 1000);
gui.add(roads, "visible").name("Roads Visible");

const buildings = new THREE.Group();
scene.add(buildings);
// NOTE: no await here to load in background
addInBatchesStreaming(buildings, loadBuildings(map), 50);
gui.add(buildings, "visible").name("Buildings Visible");

const carRoutes = new THREE.Group();
scene.add(carRoutes);
gui.add(carRoutes, "visible").name("Cars Visible");

let downLoc = { x: 0, y: 0 };
renderer.domElement.addEventListener("pointerdown", (e) => {
  downLoc = { x: e.clientX, y: e.clientY };
});
renderer.domElement.addEventListener(
  "pointerup",
  (e) => {
    if (downLoc.x !== e.clientX || downLoc.y !== e.clientY) {
      return;
    }

    const route = makeRandomCarRoute(generateVibrantColor());
    carRoutes.add(route);
  },
  false
);

renderer.setAnimationLoop(animate);
function animate() {
  update();
  controls.update();
  renderer.render(scene, camera);
}

function update() {
  for (const carRoute of carRoutes.children) {
    const car = carRoute.getObjectByName("car");
    // TODO: update
  }
}

```

```

}

window.addEventListener("resize", () => {
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(window.innerWidth, window.innerHeight);
});
}

function makeRandomCarRoute(color: THREE.Color) {
  const routeStart = randomRouteNodeId(routeMap);
  const routeEnd = randomRouteNodeId(routeMap);
  const routeNodes = findRoute(routeMap, routeStart, routeEnd);
  if (routeNodes.length === 0) {
    return makeRandomCarRoute(color);
  }

  const carAndRouteGroup = new THREE.Group();

  for (let i = 1; i < routeNodes.length; i++) {
    const startPos = routeMap.get(routeNodes[i - 1])!.pos;
    const endPos = routeMap.get(routeNodes[i])!.pos;

    const routePiece = new THREE.Mesh(
      new THREE.CapsuleGeometry(1, startPos.distanceTo(endPos)),
      new THREE.MeshBasicMaterial({
        color: color,
        transparent: true,
        opacity: 0.5,
      })
    );
    routePiece.position.copy(startPos).add(endPos).multiplyScalar(0.5);
    routePiece.lookAt(startPos);
    routePiece.rotateX(Math.PI / 2);

    carAndRouteGroup.add(routePiece);
  }

  const car = new THREE.Mesh(
    new THREE.SphereGeometry(5),
    new THREE.MeshBasicMaterial({ color })
  );
  car.name = "car";
  car.position.add(routeMap.get(routeStart)!.pos);
  carAndRouteGroup.add(car);

  return carAndRouteGroup;
}

function findRoute(
  routeMap: RouteMap,
  startId: number,
  endId: number
): number[] {
  const visited = new Set<number>();
  const queue: { id: number; path: number[]; cost: number }[] = [

```

```

    { id: startId, path: [startId], cost: 0 },
  ];

  while (queue.length > 0) {
    queue.sort((a, b) => a.cost - b.cost);
    const { id, path, cost } = queue.shift()!;

    if (id === endId) {
      return path;
    }

    if (visited.has(id)) {
      continue;
    }
    visited.add(id);

    const node = routeMap.get(id)!;
    for (const link of node.links) {
      const newPath = [...path, link.targetId];
      const newCost = cost + link.distance;
      queue.push({ id: link.targetId, path: newPath, cost: newCost });
    }
  }

  return [];
}

type RouteMap = Map<number, RouteNode>;
type RouteNode = {
  node: OSMNode;
  pos: THREE.Vector3;
  links: { distance: number; targetId: number }[];
};

function buildRouteMap(map: OSMJson): RouteMap {
  const routeMap = new Map<number, RouteNode>();
  for (const road of map.roads) {
    let prevRouteNode = null as RouteNode | null;
    for (const node of road.nodes) {
      let routeNode = routeMap.get(node.id);
      if (routeNode === undefined) {
        routeNode = { node, pos: osmToWorldPos(map, node), links: [] };
        routeMap.set(node.id, routeNode);
      }

      if (prevRouteNode !== null) {
        const existingLink = routeNode.links.find(
          (link) => link.targetId === prevRouteNode!.node.id
        );
        if (!existingLink) {
          const distance = routeNode.pos.distanceTo(prevRouteNode.pos);
          routeNode.links.push({ targetId: prevRouteNode.node.id, distance });
        }
      }

      prevRouteNode = routeNode;
    }
  }
}

```

```

    }
    return routeMap;
}

function randomRouteNodeId(routeMap: Map<number, RouteNode>) {
    const randomIndex = Math.floor(Math.random() * routeMap.size);
    const id = routeMap.keys().drop(randomIndex).take(1).toArray()[0];
    return id;
}

function* loadRoads(map: OSMJson) {
    for (const mapRoad of map.roads) {
        const points: THREE.Vector3[] = [];
        for (const node of mapRoad.nodes) {
            points.push(osmToWorldPos(map, node));
        }
        if (points.length < 2) {
            continue;
        }

        const streetGeometry = new THREE.BufferGeometry().setFromPoints(points);
        const streetMaterial = new THREE.LineBasicMaterial({
            color: "green",
            linewidth: 2,
        });
        const road = new THREE.Line(streetGeometry, streetMaterial);
        if (["motorway", "trunk"].includes(mapRoad.type)) {
            road.material.color.set("blue");
        } else if (["primary", "secondary"].includes(mapRoad.type)) {
            road.material.color.set("green");
        } else if (["service", "footway", "path"].includes(mapRoad.type)) {
            road.material.color.set("brown");
        }
        yield road;
    }
}

function* loadBuildings(map: OSMJson) {
    const material = new THREE.MeshPhongMaterial({ color: "white" });

    for (const building of map.buildings) {
        const points: THREE.Vector3[] = [];
        for (const node of building.nodes) {
            points.push(osmToWorldPos(map, node));
        }
        if (points.length < 3) {
            continue;
        }

        const shape = new THREE.Shape();
        shape.moveTo(points[0].x, points[0].z);
        for (let i = 1; i < points.length; i++) {
            shape.lineTo(points[i].x, points[i].z);
        }
        shape.lineTo(points[0].x, points[0].z);
    }
}

```

```

    const geometry = new THREE.ExtrudeGeometry(shape, {
      depth: building.height,
      bevelEnabled: false,
      steps: 1,
    });
    geometry.rotateX(Math.PI / 2);
    geometry.translate(0, building.height, 0);

    const mesh = new THREE.Mesh(geometry, material);
    yield mesh;

    const wireMaterial = new THREE.MeshPhongMaterial({
      color: building.color,
      wireframe: true,
    });
    const wireMesh = new THREE.Mesh(geometry, wireMaterial);
    yield wireMesh;
  }
}

async function addInBatchesStreaming(
  container: THREE.Group | THREE.Scene,
  generator: Generator<THREE.Object3D>,
  batchSize = 500
) {
  let batch = [];
  for (const item of generator) {
    batch.push(item);
    if (batch.length === batchSize) {
      container.add(...batch);
      batch = [];
      await new Promise(requestAnimationFrame);
    }
  }
  if (batch.length !== 0) {
    container.add(...batch);
  }
}

function osmToWorldPos(map: OSMJson, node: OSMNode) {
  const { minLat, maxLat, minLon, maxLon } = map.bounds;
  let x = ((node.lon - minLon) / (maxLon - minLon)) * 800 - 400;
  let z = -(((node.lat - minLat) / (maxLat - minLat)) * 800 - 400);
  return new THREE.Vector3(x, 0, z);
}

function generateVibrantColor() {
  const hue = Math.random();
  const saturation = Math.random() * (1 - 0.75) + 0.75;
  const lightness = Math.random() * (1 - 0.5) + 0.25;

  const color = new THREE.Color();
  color.setHSL(hue, saturation, lightness);

  return color;
}

```

Програмний код фінальної симуляції

```
import * as THREE from "three";
import { MapControls } from "three/addons/controls/MapControls.js";
import GUI from "lil-gui";

// parameters
const INTERSECTION_RADIUS = 20;
const ROAD_LENGTH = 120;
const LANE_WIDTH = 6;
const CAR_LENGTH = 8;
const CAR_WIDTH = 3;
const SPAWN_DIST = 60;
const TRAFFIC_LIGHT_CYCLE = 0.7;

type Road = {
  angle: number;
  entryPos: THREE.Vector3;
  exitPos: THREE.Vector3;
  entryDir: THREE.Vector3;
  exitDir: THREE.Vector3;
  entryQueue: Car[];
  exitQueueCount: number;
  trafficLight: TrafficLight;
  entryLabel: THREE.Sprite;
  exitLabel: THREE.Sprite;
};

type Car = THREE.Group & {
  roadIdx: number;
  targetExitIdx: number;
  state: "approaching" | "waiting" | "turning" | "exiting" | "done";
  progress: number;
  exitProgress: number;
};

type TrafficLight = {
  mesh: THREE.Mesh;
  isGreen: boolean;
};

let roads: Road[] = [];
let cars: Car[] = [];
let scene: THREE.Scene,
    camera: THREE.PerspectiveCamera,
    renderer: THREE.WebGLRenderer,
    controls: MapControls;
let gui: GUI;
let trafficLightIdx = 0;
let trafficLightTimer = 0;
```

```

let simTime = 0;
let simRunning = false;
let config = {
  intersectionType: "4-way" as "4-way" | "t-section",
  maxCars: 60,
  algorithm: "q-learning" as "sequential" | "random" | "q-learning",
  simSpeed: 100,
  randomSeed: 1,
};

// store best times and weights per seed in localStorage
type QLearningMemory = {
  [seed: string]: {
    bestTime: number;
    weights: number[];
  };
};

function getQLearningMemory(): QLearningMemory {
  try {
    return JSON.parse(localStorage.getItem("qLearningMemory") || "{}");
  } catch {
    return {};
  }
}

function setQLearningMemory(mem: QLearningMemory) {
  localStorage.setItem("qLearningMemory", JSON.stringify(mem));
}

let qLearningMemory = getQLearningMemory();
let runSeed = 0;

// --- q-learning model class ---
class QLearningModel {
  weights: number[];
  bestWeights: number[];
  bestTime: number;
  seedKey: string;

  constructor(seedKey: string, memory: QLearningMemory) {
    this.seedKey = seedKey;
    const mem = memory[seedKey];
    if (mem) {
      this.weights = this.generateNewWeights(mem.weights);
      this.bestWeights = [...mem.weights];
      this.bestTime = mem.bestTime;
    } else {
      this.weights = this.generateNewWeights([0.5, 2.0, 1.0]);
      this.bestWeights = [0.5, 2.0, 1.0];
      this.bestTime = Infinity;
    }
  }

  // generate new weights for exploration (always use math.random)
  generateNewWeights(base: number[]): number[] {
    const alpha = 0.2;
    return [

```

```

        Math.max(0, Math.min(2, base[0] + (Math.random() - 0.5) * alpha)),
        Math.max(0, Math.min(5, base[1] + (Math.random() - 0.5) * alpha)),
        Math.max(0, Math.min(3, base[2] + (Math.random() - 0.5) * alpha)),
    ];
}

// select next traffic light index based on q-learning
select(roads: Road[], simTime: number): number {
    let bestIdx = 0;
    let bestScore = -Infinity;
    for (let i = 0; i < roads.length; i++) {
        const queue = roads[i].entryQueue;
        const queueLen = queue.length;
        if (queueLen === 0) {
            continue;
        }
        let maxWait = 0;
        let totalWait = 0;
        for (const car of queue) {
            const arrivalTime = (car.userData.arrivalTime ?? 0) as number;
            const wait = simTime - arrivalTime;
            totalWait += wait;
            if (wait > maxWait) {
                maxWait = wait;
            }
        }
        const avgWait = totalWait / queueLen;
        const score =
            this.weights[0] * queueLen +
            this.weights[1] * maxWait +
            this.weights[2] * avgWait;
        if (score > bestScore) {
            bestScore = score;
            bestIdx = i;
        }
    }
    if (bestScore === -Infinity) {
        return 0;
    }
    return bestIdx;
}

// update weights after a run
update(simTime: number, memory: QLearningMemory) {
    let reward = 0;
    if (simTime < this.bestTime) {
        reward =
            this.bestTime === Infinity
                ? 1
                : (this.bestTime - simTime) / this.bestTime;
        memory[this.seedKey] = {
            bestTime: simTime,
            weights: [...this.weights],
        };
    }
    this.bestTime = simTime;
    this.bestWeights = [...this.weights];
}

```

```

    } else {
        reward =
            this.bestTime === Infinity
                ? -1
                : (this.bestTime - simTime) / this.bestTime;
    }
    this.weights = this.generateNewWeights(this.bestWeights);
}
}

// --- q-learning model instance ---
let qLearningModel: QLearningModel;

// state for simulation
let carsSpawned = 0;
let carSpawnTimer = 0;
let lastTime: number | undefined = undefined;

// --- deterministic random number generator (mulberry32) ---
let randomFn = Math.random;
function setSeed(seed: number) {
    function mulberry32(a: number) {
        return function () {
            let t = (a += 0x6d2b79f5);
            t = Math.imul(t ^ (t >>> 15), t | 1);
            t ^= t + Math.imul(t ^ (t >>> 7), t | 61);
            return ((t ^ (t >>> 14)) >>> 0) / 4294967296;
        };
    }
    randomFn = mulberry32(seed || 1);
}

qLearningMemory = getQLearningMemory();
runSeed = seed || 1;
qLearningModel = new QLearningModel(String(runSeed), qLearningMemory);
}

main();

function main() {
    gui = new GUI({ title: "Options" });
    gui
        .add(config, "intersectionType", ["4-way", "t-section"])
        .name("Intersection");
    gui
        .add(config, "maxCars", 10, 200, 1)
        .name("Total Cars")
        .onChange((v: number) => {
            config.maxCars = Math.round(v);
        });
    gui
        .add(config, "algorithm", {
            Sequential: "sequential",
            Random: "random",
            "Q-Learning": "q-learning",
        })
        .name("Algorithm");
}

```

```

gui
  .add(config, "simSpeed", 0.5, 100, 0.5)
  .name("Sim Speed")
  .onChange((v: number) => {
    config.simSpeed = v;
  });
gui
  .add(config, "randomSeed", 0, 999999, 1)
  .name("Random Seed")
  .onChange(() => {
    setSeed(config.randomSeed);
    updateWeightsDisplay();
  });
gui.add({ start: startSimulation }, "start").name("Start Simulation");

setSeed(config.randomSeed);

createWeightsDisplay();

const canvas = document.querySelector("canvas")! as HTMLCanvasElement;

scene = new THREE.Scene();
scene.background = new THREE.Color("skyblue");

renderer = new THREE.WebGLRenderer({ canvas, antialias: true });
renderer.setPixelRatio(window.devicePixelRatio);
renderer.setSize(window.innerWidth, window.innerHeight);
renderer.setAnimationLoop(animate);

camera = new THREE.PerspectiveCamera(
  60,
  window.innerWidth / window.innerHeight,
  1,
  2000
);
camera.position.set(0, 180, 0);
camera.lookAt(0, 0, 0);

controls = new MapControls(camera, renderer.domElement);
controls.enableDamping = true;
controls.dampingFactor = 0.1;
controls.maxPolarAngle = Math.PI / 2;

// lighting
const light = new THREE.HemisphereLight("white", "gray", 1.2);
light.position.set(0, 500, 0);
scene.add(light);

createIntersection();

setSeed(config.randomSeed);

window.addEventListener("resize", onWindowResize);
}

function createWeightsDisplay() {

```

```

let weightsDiv = document.getElementById("weights-display");
if (!weightsDiv) {
    weightsDiv = document.createElement("div");
    weightsDiv.id = "weights-display";
    weightsDiv.style.position = "fixed";
    weightsDiv.style.bottom = "60px";
    weightsDiv.style.right = "20px";
    weightsDiv.style.fontSize = "1.2em";
    weightsDiv.style.color = "#fff";
    weightsDiv.style.background = "rgba(0,0,0,0.5)";
    weightsDiv.style.padding = "8px 16px";
    weightsDiv.style.borderRadius = "8px";
    weightsDiv.style.zIndex = "1000";
    document.body.appendChild(weightsDiv);
}
updateWeightsDisplay();
}

function updateWeightsDisplay() {
    let weightsDiv = document.getElementById("weights-display");
    if (!weightsDiv) {
        return;
    }
    weightsDiv.innerHTML =
        `Q-Learning Weights</b><br>` +
        `Current: ` +
        `[${qLearningModel.weights.map((w) => w.toFixed(3)).join(", ")}]<br>` +
        `Best: ` +
        `[${qLearningModel.bestWeights.map((w) => w.toFixed(3)).join(", ")}]<br>` +
        `Best time for seed ${runSeed}: ` +
        (qLearningModel.bestTime < Infinity
            ? qLearningModel.bestTime.toFixed(2) + "s"
            : "N/A");
}

// --- traffic light algorithm functions ---
function selectSequentialLight(currentIdx: number, roads: Road[]): number {
    return (currentIdx + 1) % roads.length;
}

function selectRandomLight(currentIdx: number, roads: Road[]): number {
    let nextIdx;
    do {
        nextIdx = Math.floor(randomFn() * roads.length);
    } while (nextIdx === currentIdx && roads.length > 1);
    return nextIdx;
}

function selectQLearningLight(
    roads: Road[],
    simTime: number,
    currentIdx: number
): number {
    const idx = qLearningModel.select(roads, simTime);
    if (roads[idx].entryQueue.length === 0) {
        return selectSequentialLight(currentIdx, roads);
    }
}

```

```

    }
    return idx;
}

function startSimulation() {
    simRunning = false;
    for (const obj of [...scene.children]) {
        if (!(obj instanceof THREE.Camera) && !(obj instanceof THREE.Light)) {
            scene.remove(obj);
        }
    }
    const timerDiv = document.getElementById("sim-timer");
    if (timerDiv) {
        timerDiv.remove();
    }

    cars = [];
    roads = [];
    trafficLightIdx = 0;
    trafficLightTimer = 0;
    simTime = 0;
    carsSpawned = 0;
    carSpawnTimer = 0;
    lastTime = undefined;

    setSeed(config.randomSeed);

    if (config.algorithm === "q-learning") {
        qLearningModel.weights = qLearningModel.generateNewWeights(
            qLearningModel.bestWeights
        );
        updateWeightsDisplay();
    }

    createIntersection();
    for (let i = 0; i < roads.length; i++) {
        roads[i].trafficLight.isGreen = false;
        roads[i].trafficLight.mesh.material.color.set("red");
    }
    if (roads.length > 0) {
        roads[0].trafficLight.isGreen = true;
        roads[0].trafficLight.mesh.material.color.set("green");
    }

    setTimeout(() => {
        simRunning = true;
    }, 0);
}

function createIntersection() {
    let angles: number[] = [];
    if (config.intersectionType === "4-way") {
        angles = [0, Math.PI / 2, Math.PI, (3 * Math.PI) / 2];
    } else if (config.intersectionType === "t-section") {
        angles = [0, Math.PI / 2, Math.PI];
    }
}

```

```

const CENTER_RADIUS = INTERSECTION_RADIUS + LANE_WIDTH * 1.2;

const center = new THREE.Mesh(
  new THREE.CylinderGeometry(CENTER_RADIUS, CENTER_RADIUS, 1, 40),
  new THREE.MeshPhongMaterial({ color: "gray" })
);
center.position.y = 0.5;
scene.add(center);

roads = [];
for (let i = 0; i < angles.length; i++) {
  const angle = angles[i];
  const dir = new THREE.Vector3(Math.cos(angle), 0, Math.sin(angle));
  const laneOffset = dir
    .clone()
    .cross(new THREE.Vector3(0, 1, 0))
    .setLength(LANE_WIDTH * 0.6);

  const entryLaneCenter = dir
    .clone()
    .multiplyScalar(CENTER_RADIUS + ROAD_LENGTH / 2)
    .add(laneOffset);
  const entryGeom = new THREE.BoxGeometry(ROAD_LENGTH, 1, LANE_WIDTH);
  const entryMat = new THREE.MeshPhongMaterial({ color: 0x333333 });
  const entryMesh = new THREE.Mesh(entryGeom, entryMat);
  entryMesh.position.copy(entryLaneCenter);
  entryMesh.position.y = 0.5;
  entryMesh.rotation.y = -angle;
  scene.add(entryMesh);

  const exitLaneCenter = dir
    .clone()
    .multiplyScalar(CENTER_RADIUS + ROAD_LENGTH / 2)
    .sub(laneOffset);
  const exitGeom = new THREE.BoxGeometry(ROAD_LENGTH, 1, LANE_WIDTH);
  const exitMat = new THREE.MeshPhongMaterial({ color: 0x222222 });
  const exitMesh = new THREE.Mesh(exitGeom, exitMat);
  exitMesh.position.copy(exitLaneCenter);
  exitMesh.position.y = 0.5;
  exitMesh.rotation.y = -angle;
  scene.add(exitMesh);

  const dividerGeom = new THREE.BoxGeometry(ROAD_LENGTH, 0.2, 0.2);
  const dividerMat = new THREE.MeshBasicMaterial({ color: 0xffffff });
  const dividerMesh = new THREE.Mesh(dividerGeom, dividerMat);
  dividerMesh.position.copy(
    dir.clone().multiplyScalar(CENTER_RADIUS + ROAD_LENGTH / 2)
  );
  dividerMesh.position.y = 1.1;
  dividerMesh.rotation.y = -angle;
  scene.add(dividerMesh);

  const lightGeom = new THREE.SphereGeometry(2.2, 16, 16);
  const lightMat = new THREE.MeshPhongMaterial({ color: "red" });
  const lightMesh = new THREE.Mesh(lightGeom, lightMat);

```

```

    lightMesh.position.copy(
      dir
        .clone()
        .multiplyScalar(CENTER_RADIUS + 8)
        .add(laneOffset)
    );
    lightMesh.position.y = 6;
    scene.add(lightMesh);

    const entryLabel = makeLabelSprite("0", "#3af");
    entryLabel.position.copy(
      dir
        .clone()
        .multiplyScalar(CENTER_RADIUS + 18)
        .add(laneOffset)
    );
    entryLabel.position.y = 16;
    scene.add(entryLabel);

    const exitLabel = makeLabelSprite("0", "#fa3");
    exitLabel.position.copy(
      dir
        .clone()
        .multiplyScalar(CENTER_RADIUS + 18)
        .sub(laneOffset)
    );
    exitLabel.position.y = 16;
    scene.add(exitLabel);

    roads.push({
      angle,
      entryPos: entryLaneCenter,
      exitPos: exitLaneCenter,
      entryDir: dir,
      exitDir: dir,
      entryQueue: [],
      exitQueueCount: 0,
      trafficLight: { mesh: lightMesh, isGreen: false },
      entryLabel,
      exitLabel,
    });
  }
  roads[0].trafficLight.isGreen = true;
  roads[0].trafficLight.mesh.material.color.set("green");
}

function animate() {
  let now = performance.now();
  if (lastTime === undefined) {
    lastTime = now;
  }
  const deltaRaw = (now - lastTime) / 1000;
  lastTime = now;
  const delta = deltaRaw * (config.simSpeed || 1);

  if (simRunning) {

```

```

    simTime += delta;
}

if (simRunning) {
    trafficLightControl(delta);
}

if (simRunning) {
    const SPAWN_INTERVAL = 0.6 / (config.simSpeed || 1);

    carSpawnTimer += deltaRaw;
    while (carsSpawned < config.maxCars && carSpawnTimer >= SPAWN_INTERVAL) {
        spawnCar();
        carsSpawned++;
        carSpawnTimer -= SPAWN_INTERVAL;
    }
}

for (let i = cars.length - 1; i >= 0; i--) {
    const car = cars[i];
    updateCar(car, delta);
    if (car.state === "done") {
        scene.remove(car);
        cars.splice(i, 1);
    }
}

for (let i = 0; i < roads.length; i++) {
    roads[i].entryLabel.material.map = makeLabelTexture(
        roads[i].entryQueue.length.toString(),
        "#3af"
    );
    roads[i].exitLabel.material.map = makeLabelTexture(
        roads[i].exitQueueCount.toString(),
        "#fa3"
    );
    roads[i].entryLabel.material.map.needsUpdate = true;
    roads[i].exitLabel.material.map.needsUpdate = true;
}

drawSimTimer();

if (
    simRunning &&
    cars.length === 0 &&
    carsSpawned >= config.maxCars &&
    simTime > 0
) {
    simRunning = false;

    if (config.algorithm === "q-learning") {
        qLearningModel.update(simTime, qLearningMemory);
        setQLearningMemory(qLearningMemory);
        updateWeightsDisplay();
    }
}
}

```

```

controls.update();
renderer.render(scene, camera);
}

function drawSimTimer() {
  let timerDiv = document.getElementById("sim-timer");
  if (!timerDiv) {
    timerDiv = document.createElement("div");
    timerDiv.id = "sim-timer";
    timerDiv.style.position = "fixed";
    timerDiv.style.bottom = "10px";
    timerDiv.style.right = "20px";
    timerDiv.style.fontSize = "2em";
    timerDiv.style.color = "#fff";
    timerDiv.style.textShadow = "0 0 8px #000";
    timerDiv.style.zIndex = "1000";
    document.body.appendChild(timerDiv);
  }
  const carsPassed = carsSpawned - cars.length;
  const progress = config.maxCars > 0 ? carsPassed / config.maxCars : 0;
  let status = simRunning
    ? `Time: ${simTime.toFixed(1)}s | Progress: ${(progress * 100).toFixed(
      1
    )}% (${carsPassed}/${config.maxCars})`
    : carsSpawned > 0
    ? `Finished in ${simTime.toFixed(1)}s | Progress: 100% (${config.maxCars}/${
      config.maxCars
    })`
    : "Press Start Simulation";
  timerDiv.textContent = status;
}

// traffic light control function (customizable)
function trafficLightControl(delta: number) {
  trafficLightTimer += delta;
  if (trafficLightTimer > TRAFFIC_LIGHT_CYCLE) {
    roads[trafficLightIdx].trafficLight.isGreen = false;
    roads[trafficLightIdx].trafficLight.mesh.material.color.set("red");

    if (config.algorithm === "sequential") {
      trafficLightIdx = selectSequentialLight(trafficLightIdx, roads);
    } else if (config.algorithm === "random") {
      trafficLightIdx = selectRandomLight(trafficLightIdx, roads);
    } else if (config.algorithm === "q-learning") {
      trafficLightIdx = selectQLearningLight(roads, simTime, trafficLightIdx);
    }

    roads[trafficLightIdx].trafficLight.isGreen = true;
    roads[trafficLightIdx].trafficLight.mesh.material.color.set("green");
    trafficLightTimer = 0;

    const entryQueue = roads[trafficLightIdx].entryQueue;
    for (const car of entryQueue) {
      if (car.state === "waiting") {
        car.state = "turning";
      }
    }
  }
}

```

```

        car.progress = 0;
    }
}
}

function spawnCar() {
    let entryIdx = Math.floor(randomFn() * roads.length);
    let exitIdx;
    do {
        exitIdx = Math.floor(randomFn() * roads.length);
    } while (exitIdx === entryIdx);

    const entryRoad = roads[entryIdx];
    const car = makeCar(entryIdx, exitIdx);

    const laneOffset = entryRoad.entryDir
        .clone()
        .cross(new THREE.Vector3(0, 1, 0))
        .setLength(LANE_WIDTH * 0.6);
    car.position.copy(
        entryRoad.entryDir
            .clone()
            .multiplyScalar(
                INTERSECTION_RADIUS +
                LANE_WIDTH * 1.2 +
                ROAD_LENGTH / 2 +
                SPAWN_DIST +
                CAR_LENGTH * entryRoad.entryQueue.length
            )
            .add(laneOffset)
    );
    car.position.y = 2;
    car.rotation.y = Math.atan2(entryRoad.entryDir.x, entryRoad.entryDir.z);
    car.userData.lane = "entry";
    scene.add(car);
    cars.push(car);
    entryRoad.entryQueue.push(car);
    roads[exitIdx].exitQueueCount++;
}

function makeCar(roadIdx: number, targetExitIdx: number): Car {
    const car = new THREE.Group() as Car;
    car.roadIdx = roadIdx;
    car.targetExitIdx = targetExitIdx;
    car.state = "approaching";
    car.progress = 0;
    car.exitProgress = 0;

    const mesh = new THREE.Mesh(
        new THREE.BoxGeometry(CAR_WIDTH, 3, CAR_LENGTH),
        new THREE.MeshPhongMaterial({ color: generateVibrantColor() })
    );
    mesh.position.y = 0;
    car.add(mesh);
}

```

```

    return car;
}

function updateCar(car: Car, delta: number) {
    const entryRoad = roads[car.roadIdx];
    const exitRoad = roads[car.targetExitIdx];
    const CENTER_RADIUS = INTERSECTION_RADIUS + LANE_WIDTH * 1.2;

    const speed = 90;

    if (car.state === "approaching") {
        const queueIdx = entryRoad.entryQueue.indexOf(car);
        const laneOffset = entryRoad.entryDir
            .clone()
            .cross(new THREE.Vector3(0, 1, 0))
            .setLength(LANE_WIDTH * 0.6);
        const stopDist = CENTER_RADIUS + CAR_LENGTH * (queueIdx + 0.5);
        const targetPos = entryRoad.entryDir
            .clone()
            .multiplyScalar(stopDist)
            .add(laneOffset);
        const toTarget = targetPos.clone().sub(car.position);
        const dist = toTarget.length();
        if (dist > 0.1) {
            car.position.add(
                toTarget.normalize().multiplyScalar(Math.min(speed * delta, dist))
            );
            car.rotation.y = Math.atan2(entryRoad.entryDir.x, entryRoad.entryDir.z);
        } else {
            car.position.copy(targetPos);
            car.rotation.y = Math.atan2(entryRoad.entryDir.x, entryRoad.entryDir.z);
            car.state = "waiting";
            car.userData.lane = "entry";
            if (typeof car.userData.arrivalTime === "undefined") {
                car.userData.arrivalTime = simTime;
            }
        }
    } else if (car.state === "waiting") {
        // handled in animate() when light turns green for all waiting cars
    } else if (car.state === "turning") {
        car.progress += delta * 0.45;
        const t = Math.min(car.progress, 1);

        const entryLaneOffset = entryRoad.entryDir
            .clone()
            .cross(new THREE.Vector3(0, 1, 0))
            .setLength(LANE_WIDTH * 0.6);
        const exitLaneOffset = exitRoad.exitDir
            .clone()
            .cross(new THREE.Vector3(0, 1, 0))
            .setLength(-LANE_WIDTH * 0.6);

        const p0 = entryRoad.entryDir
            .clone()
            .multiplyScalar(CENTER_RADIUS)
            .add(entryLaneOffset);
    }
}

```

```

const p1 = new THREE.Vector3(0, 0, 0);
const p2 = exitRoad.exitDir
    .clone()
    .multiplyScalar(CENTER_RADIUS)
    .add(exitLaneOffset);
const pos = bezier2(p0, p1, p2, t);
car.position.copy(pos);
car.position.y = 2;
const tangent = bezier2Tangent(p0, p1, p2, t);
car.rotation.y = Math.atan2(tangent.x, tangent.z);

if (t >= 1) {
    car.state = "exiting";
    car.exitProgress = 0;
    const exitStart = exitRoad.exitDir
        .clone()
        .multiplyScalar(CENTER_RADIUS)
        .add(exitLaneOffset);
    car.position.copy(exitStart);
    car.rotation.y = Math.atan2(exitRoad.exitDir.x, exitRoad.exitDir.z);
    car.userData.lane = "exit";
    const idx = entryRoad.entryQueue.indexOf(car);
    if (idx !== -1) {
        entryRoad.entryQueue.splice(idx, 1);
    }
    exitRoad.exitQueueCount--;
}
} else if (car.state === "exiting") {
    const exitLaneOffset = exitRoad.exitDir
        .clone()
        .cross(new THREE.Vector3(0, 1, 0))
        .setLength(-LANE_WIDTH * 0.6);
    const start = exitRoad.exitDir
        .clone()
        .multiplyScalar(CENTER_RADIUS)
        .add(exitLaneOffset);
    const end = exitRoad.exitDir
        .clone()
        .multiplyScalar(CENTER_RADIUS + ROAD_LENGTH / 2 + SPAWN_DIST)
        .add(exitLaneOffset);
    const totalDist = end.clone().sub(start).length();
    car.exitProgress += (speed * delta) / totalDist;
    const t = Math.min(car.exitProgress, 1);
    const pos = start.clone().lerp(end, t);
    car.position.copy(pos);
    car.position.y = 2;
    car.rotation.y = Math.atan2(exitRoad.exitDir.x, exitRoad.exitDir.z);
    if (t >= 1) {
        car.state = "done";
    }
}
}
}

function bezier2(
    p0: THREE.Vector3,
    p1: THREE.Vector3,

```

```

    p2: THREE.Vector3,
    t: number
  ) {
    return p0
      .clone()
      .multiplyScalar((1 - t) * (1 - t))
      .add(p1.clone().multiplyScalar(2 * (1 - t) * t))
      .add(p2.clone().multiplyScalar(t * t));
  }

function bezier2Tangent(
  p0: THREE.Vector3,
  p1: THREE.Vector3,
  p2: THREE.Vector3,
  t: number
) {
  return p1
    .clone()
    .sub(p0)
    .multiplyScalar(2 * (1 - t))
    .add(
      p2
        .clone()
        .sub(p1)
        .multiplyScalar(2 * t)
    )
    .normalize();
}

function makeLabelSprite(text: string, color: string) {
  const map = makeLabelTexture(text, color);
  const material = new THREE.SpriteMaterial({ map, depthTest: false });
  const sprite = new THREE.Sprite(material);
  sprite.scale.set(18, 8, 1);
  return sprite;
}

function makeLabelTexture(text: string, color: string = "white") {
  const canvas = document.createElement("canvas");
  canvas.width = 128;
  canvas.height = 64;
  const ctx = canvas.getContext("2d")!;
  ctx.clearRect(0, 0, canvas.width, canvas.height);
  ctx.font = "bold 48px sans-serif";
  ctx.textAlign = "center";
  ctx.textBaseline = "middle";
  ctx.fillStyle = color;
  ctx.strokeStyle = "black";
  ctx.lineWidth = 6;
  ctx.strokeText(text, canvas.width / 2, canvas.height / 2);
  ctx.fillText(text, canvas.width / 2, canvas.height / 2);
  return new THREE.CanvasTexture(canvas);
}

function generateVibrantColor() {
  const hue = randomFn();

```

```
const saturation = randomFn() * (1 - 0.75) + 0.75;
const lightness = randomFn() * (1 - 0.25) + 0.25;
const color = new THREE.Color();
color.setHSL(hue, saturation, lightness);
return color;
}

function onWindowResize() {
  camera.aspect = window.innerWidth / window.innerHeight;
  camera.updateProjectionMatrix();
  renderer.setSize(window.innerWidth, window.innerHeight);
}
```



Звіт подібності

метадані

Назва організації

Kyiv National Economic University named after Vadym Hetman KNEU

Заголовок

Розроблення системи автономного управління дорожнього руху на перехрестях

Автор

Науковий керівник / Експерт

Башкатов Гліб РуслановичТішков Б.О.

підрозділ

кафедра інформаційних систем в економіці

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.

**25**

Довжина фрази для коефіцієнта подібності 2

**9337**

Кількість слів

**73762**

Кількість символів

Bashkatov H. Exploring possibilities of data analysis and transformation using lightweight web-based tools. Proceedings of the XI International Scientific and Practical Conference. Seville, Spain. 2024. Pp. 24-25

URL: <https://isg-konf.com/modern-generation-current-problems-experience-development-prospects/>

Exploring Possibilities of Data Analysis and Transformation Using Lightweight Web-Based Tools

Introduction. The growing need for efficient, accessible data analysis tools has sparked interest in browser-based solutions that can simplify workflows. Traditional methods, such as Jupyter Notebooks or full-scale desktop IDEs, often require significant setup and feel excessive for quick, exploratory tasks. While Jupyter offers strong interactivity, its Python-centric nature and setup requirements can be restrictive. This exploration seeks to understand how a lightweight, JavaScript-based approach within a familiar, web-based code editing environment could streamline data transformation without overcomplicating the process.

Objective. The objective is to investigate how a VS Code for the Web extension could facilitate data analysis and transformation using JavaScript. The focus is on understanding whether such a tool could simplify common tasks and enhance the flexibility of working in-browser, all while avoiding the need for extensive configurations or heavy software installations.

Features and Capabilities. The exploration will cover essential capabilities, such as: transforming complex or poorly structured Excel data into formats like JSON or iCal for exporting making calendars; using JavaScript to map, filter, and manipulate CSV or plain text structured data for varied purposes; performing very specific text processing, such as reformatting or extracting data subsets. These functionalities aim to bridge the gap between basic text editors and more complex tools like Jupyter or desktop IDEs, turning VS Code for the Web into an experimental space for data handling.

Integrated Workflow and Usability. This investigation will take a closer look at how VS Code for the Web's advanced text editing features—such as multi-cursor support, keyboard shortcuts, and efficient file navigation—can improve productivity. The emphasis will be on whether the familiar and user-friendly interface of VS Code can elevate productivity without requiring users to invest time in learning highly specialized commands, as is often the case with tools like VIM or Emacs. Intuitive shortcuts, such as selecting words with Ctrl + Shift + Arrow or using Shift + Home/End to highlight lines, are universally recognized across many software applications also work in VS Code, reducing the learning curve for users and improving editing speed [1]. Unlike environments with a steep learning curve, these features in VS Code are designed to be accessible yet powerful enough to perform basic data filtering and transformation tasks using standard techniques like search and replace. Furthermore, the integration of customizable keyboard shortcuts in VS Code provides a significant productivity boost. Studies indicate that adopting keyboard-driven workflows can enhance the efficiency of repetitive tasks,

making data transformation workflows less time-consuming. The design philosophy behind VS Code encourages flexibility and ease of use, allowing users to harness more advanced functionality without needing to learn a completely new system. This balance between power and simplicity positions VS Code for the Web as an ideal tool for lightweight, exploratory data work, with many productivity benefits over traditional mouse-based interactions.

Flexibility and Temporary Nature. This approach acknowledges that a lightweight, web-based setup isn't a comprehensive "batteries included" solution. Instead, it encourages a pragmatic use of external tools for one-off tasks—like using a base64 decoding website or an online unzipping service when that's faster and more convenient. However, thanks to JavaScript's support for HTTP imports, users can easily extend the core capabilities by incorporating specialized libraries as needed. The `math.js` library is one of the many JavaScript libraries that contain multiple helpful functions to make Data Analysis very easy with JavaScript [2]. This adaptability ensures that workflows remain flexible and efficient, empowering users to tailor their setup based on task requirements without being bogged down by unnecessary features or complexity.

Comparisons to Existing Solutions. Jupyter notebook is the most go-to platform when it comes to performing data analysis or performing data science-intense data pre-processing, EDA and data visualizations [3]. However, this project will evaluate whether a browser-based setup might fit just be enough. Unlike Python-focused environments that require a dedicated setup, this research will assess whether a simpler, more dynamic option using JavaScript can address the same needs. The comparison will also consider the advantages and limitations compared to traditional desktop IDEs.

Conclusion. This thesis explores the potential of a lightweight, web-based approach to data transformation using JavaScript within VS Code for the Web. The goal is not to create a final product but to bring together different components—JavaScript's flexibility, browser-based accessibility, and the efficient editing features of VS Code for the Web—to evaluate their combined effectiveness. The outcomes will contribute to a broader understanding of how simple, web-based tools can support and enhance data analysis workflows.

References

1. Mastering the Art of Efficiency: The Most Useful Keyboard Shortcuts to Boost Your Productivity - <https://medium.com/@shadowcasteryt/mastering-the-art-of-efficiency-the-most-useful-keyboard-shortcuts-to-boost-your-productivity-5695e5a8c786>.
2. How to Use JavaScript for Data Analysis – A Beginner's Guide. URL: <https://www.freecodecamp.org/news/basics-of-data-analysis-with-javascript>.
3. Leveraging the Power of Jupyter Notebooks. URL: <https://towardsdatascience.com/leveraging-the-power-of-jupyter-notebooks-26b4b8d7c622>.