ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ІНФОРМАЦІЙНИХ ТЕХНОЛОГІЙ
КАФЕДРА КОМП'ЮТЕРНИХ НАУК

КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Розробка програми моделювання руху по дорогам та перехрестям на мові Java»

на здобуття освітнього ступеня бакалавра
зі спеціальності 122 Комп'ютерні науки
(код, найменування спеціальності)
освітньо-професійної програми Комп'ютерні науки
(назва)

Кваліфікаційна робота містить результати власних досліджень. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

_____ Максим МОРХОВСЬКИЙ
(підпис) *(Ім'я, ПРІЗВИЩЕ здобувача)*

Виконав:
здобувач вищої освіти Максим МОРХОВСЬКИЙ
група КНД-42

Керівник: Ільїн Олег Олександрович
науковий ступінь, вчене звання

Рецензент:
науковий ступінь, вчене звання
(Ім'я, ПРІЗВИЩЕ)

Київ 2024

ДЕРЖАВНИЙ УНІВЕРСИТЕТ
ІНФОРМАЦІЙНО-КОМУНІКАЦІЙНИХ ТЕХНОЛОГІЙ

Навчально-науковий інститут інформаційних технологій

Кафедра Комп'ютерних наук

Ступінь вищої освіти Бакалавр

Спеціальність Комп'ютерні науки

Освітньо-професійна програма Комп'ютерні науки

ЗАТВЕРДЖУЮ
Завідувач кафедрою Комп'ютерних наук

_____ Віктор ВИШНІВСЬКИЙ
«_____» _____ 2024 р.

ЗАВДАННЯ
НА КВАЛІФІКАЦІЙНУ РОБОТУ

Морховському Максиму Сергійовичу
(прізвище, ім'я, по батькові здобувача)

1. Тема кваліфікаційної роботи: «Розробка програми моделювання руху по дорогам та перехрестям на мові Java»

керівник кваліфікаційної роботи Олег Ільїн д.т.н., професор,
(Ім'я, ПРІЗВИЩЕ науковий ступінь, вчене звання)
затверджені наказом Державного університету інформаційно-комунікаційних технологій від «27» 02.2024р. №36

2. Строк подання кваліфікаційної роботи «31» травня 2024р.

3. Вихідні дані до кваліфікаційної роботи: науково-технічна література з питань, що пов'язані з моделюванням руху по дорогам та перехрестям.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
Аналіз предметної області;

Розгляд існуючих методів моделювання;

Вибір технології розробки та інструментальних засобів;

Розробка програмного забезпечення.

5. Перелік графічного матеріалу

Слайди з презентації

Результат роботи створеної програми

6. Дата видачі завдання «23» лютого 2024 р.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Підбір науково-технічної літератури	23.02 – 25.02.24	
2	Дослідження поставленої задачі	24.02 – 25.02.24	
3	Аналіз методів розв’язання задачі	26.02 – 29.02.24	
4	Застосування алгоритмів на практиці	01.03 – 04.03.24	
5	Аналіз отриманих результатів	05.03 – 10.03.24	
6	Розробка програмного застосунку	12.03 – 25.03.24	
7	Написання вступу, висновків, реферату	27.03 – 30.03.24	
8	Написання основних розділів пояснювальної записки	01.04 – 16.04.24	
9	Попередній захист роботи	21.05.24	

Здобувач вищої освіти _____ Максим МОРХОВСЬКИЙ
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

Керівник
кваліфікаційної роботи _____ Олег Ільїн
(підпис) *(Ім’я, ПРІЗВИЩЕ)*

РЕФЕРАТ

Текстова частина бакалаврської роботи: 61 с., 2 табл., 23 рис., 10 джерел.

Наукове завдання – розробка програми для моделювання руху на мові Java та дослідження доступних методів симуляції дорожнього руху.

Мета роботи – проаналізувати існуючі методи візуалізації автомобільного руху та реалізувати подібну програму мовою Java.

Об`єкт дослідження – програми моделювання руху.

Предмет дослідження – програма візулазізації проїзду перехрестя на мові Java.

Сфера застосування – симуляція, навігація.

Короткий зміст роботи:

Проведено аналіз сучасних технологій візуалізації проїзду перехресть.

Розглянуто проблеми використання популярних рішень у сфері моделювання.

Розроблено програму для симуляції проїзду перехрестя на мові Java. Проведено тести роботи програми з урахуванням різних параметрів.

КЛЮЧОВІ СЛОВА: СИМУЛЯЦІЯ, ВІЗУАЛІЗАЦІЯ, ТРАФІК, ДОРОЖНІЙ РУХ.

ЗМІСТ

ВСТУП

Зростаюча необхідність у добре налагодженій мобільності є одним з перших вимог нашого часую - а разом з нею і вимоги до надійних та відповідальних засобів транспортування. Ми усі знайомі з проблемами, які з'явились з перенавантаженими транспортними мережами: затори на дорогах, аварії, забруднення повітря та шум це найпоширеніші проблеми поганої регуляції трафіку у великих містах.

Наша мобільність та системи транспортування з часом стають все складніше. Тож робити їх більш ефективними та надійними у свою чергу стає все більшим викликом. Які є найкращі методи зменшити кількість викидів у повітря? Як можна забезпечити безпеку для усіх учасників дорожнього руху? Що вибрати між кільцевою розв'язкою та адаптивним світлофором щоб забезпечити хороший транспортний потік? Як узгодити роботу громадського транспорту з безперебійним рухом на дорозі? Який вплив на транспортну мережу матимуть нові технології, такі як самохідні машини?

Симуляція трафіку це важливий інструмент для дослідження цих питань. Вона дозволяє планувальникам продуктивніше використовувати доступні ресурси та бюджет при розширенні чи реконструкції транспортних систем.

Актуальність: Симуляційні моделі допомагають зрозуміти вплив різних заходів та обставин на обсяг і потік транспорту за різних обставин. Отже, симуляція руху будує надійну основу для прийняття хороших і економічно ефективних рішень, роблячи рух і мобільність безпечнішими, справедливими та стійкими. Одним словом, це допоможе створити мобільність, орієнтовану на майбутнє.

Мета роботи: дослідити, чи можна створити симуляцію дорожнього руху використовуючи мову програмування Java та наскільки це буде ефективно. Основними завданнями є реалізація алгоритмів моделювання руху, створення зручного інтерфейсу користувача для введення параметрів моделювання та спостереження за його результатами, а також оцінка ефективності та швидкості розробленої програми.

1 ОСНОВНІ ВИЗНАЧЕННЯ ТА ОГЛЯД ІСНУЮЧИХ МЕТОДІВ

1.1 Основні поняття симуляції руху

Симуляція руху - це віртуальна копія реальних сценаріїв, відтворена у цифровому середовищі. У цьому середовищі проектувальники зможуть моделювати роз'їзди, коридори та практично будь-яку інфраструктуру – від доріг до розв'язок і транспортних вузлів. Як і архітектурна модель, симуляція транспорту допомагає краще зрозуміти складну систему. У містах пішоходи, велосипедисти, громадський транспорт та моторизовані засоби взаємодіють між собою у дуже вузькому просторі. І мультимодальні транспортні системи складаються не лише з різних типів учасників доріг та видів транспорту. Такі аспекти, як зміна поведінки руху або вихід на ринок нових послуг, наприклад самохідних машин, також мають значення. Симуляційні моделі дозволяють планувальникам дорожнього руху зрозуміти цю надзвичайно складну та динамічну сферу, та розробити ефективні стратегії вирішення поточних і майбутніх проблем (Рисунок 1.1).

Рисунок 1.1 - Приклад складної програми симуляції руху на прилеглій дорозі

Принципово, симуляційні моделі зосереджені на трьох вихідних значеннях для вирішення проблем з трафіком:

-перше значення- це транспортний потік. У потоках можуть бути альтернативні маршрути, визначені за кількістю транспортних засобів. Використовуючи симуляцію, планувальник може придумати, як зменшити рівень завантаженості доріг.

-другим значенням є елементи транспортної мережі. Елементи мережі у симуляції складаються з вузлів, злиття, перехрестя доріг та інших елементів дороги. Це відноситься до геометричного компонування вулиці. Використовуючи відповідне програмне забезпечення можна змінити геометричний дизайн вулиць в залежності від поточної ситуації трафіку.

-третє значення підпадає під категорію «зменшення». Симуляція може допомогти оцінити час та кошти на поїздку. Це особливо допомагає, коли оцінка покращень трафіку потребує виміру. Планувальник може зрівняти показники ефективності не витрачаючи на це додаткові кошти та час.

1.2 Класифікація моделей

Імітаційне (симуляційне) моделювання – це метод дослідження, при якому система, що вивчається, замінюється моделлю, яка з достатньою точністю описує реальну систему (побудована модель описує процеси так, як вони проходили б насправді), з якоюпроводяться есперименти з метою отримання інформації про цю систему.

Симуляційне моделювання руху дорожньо-транспортних систем виконує функцію відтворення різних сценаріїв ситуацій, яка можуть статися на дорозі для подальшого аналізу.

Існує три основні класифікації моделювання руху:

- мікроскопічне;
- макроскопічне;
- мезоскопічне.

1.2.1 Мікроскопічний метод моделювання

Мікроскопічний метод моделювання будується на основі характеристик руху різних типів транспорту у потоці, таких як автомобілі, автобуси мотоцикли тощо. Мікроскопічне моделювання спрямоване на збір параметрів даних, таких як потік, щільність руху, швидкість, час подорожі та затримки, великі черги, зупинки, споживання палива та ударні хвилі. Характеристики мікроскопічного методу моделювання були створені на основі моделі руху автомобіля у колоні, моделі зміни смуги руху та розривів між окремими водіями.

Модель руху у колоні

Ці алгоритми представили концепцію водія, який розпізнає та слідує за головною машиною на меншій швидкості. Це також може бути описане в ситуаціях руху у колонні без можливості зміни смуги. Загалом, алгоритм машини, яка їде слідом, визначає відносини з головною машиною. Чим ближче автомобіль до наступного, тим більш чутливою є реакція водія. Швидкість також впливає на чутливість реакції. Отже, якщо головна машина їде на зниженій швидкості, машинам позаду також доведеться знизити свою швидкість, що часто призводить до скупчень на магістралях та заторів.

На Рисунку 1.2 лінія шляху (h) це горизонтальна дистанція між двома автомобілями. Інтервали (S) це відстань від заднього бамперу машини у колонні до заднього бамперу головної машини яка проходить у певний момент в межах інтервалу часу. Цей інтервал визначається як різниця у часі між послідовними перетинами транспортних засобів в певній точці. *Мікроскопічне моделювання* основане на русі окремих транспортних засобів та їх положенні відносно часу та простору.

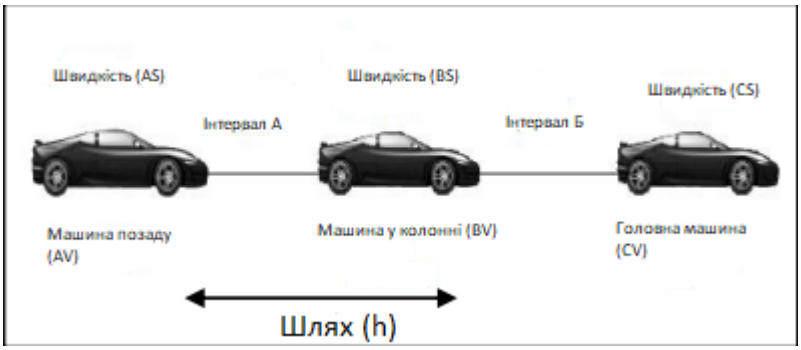


Рисунок 1.2 - Модель руху у лінії

Модель зміни смуги руху

Британський математик Пітер Гріпс запропонував розробку моделі зміни смуги руху. Це процес рішень для оцінки поведінки водія під час зміни смуги руху протягом зазначеного часу.

Зміни смуги можна розділити на два типи (Рисунок 1.3):

-обов'язкова, коли водій змінює смугу на визначену іншу смугу, наприклад, водій перестроївся в праву смугу, коли він хотів повернути праворуч на наступному перехресті;

-довільна, коли водій виїхав на сусідню смугу щоб уникнути слідування за вантажівками і збільшити швидкість.

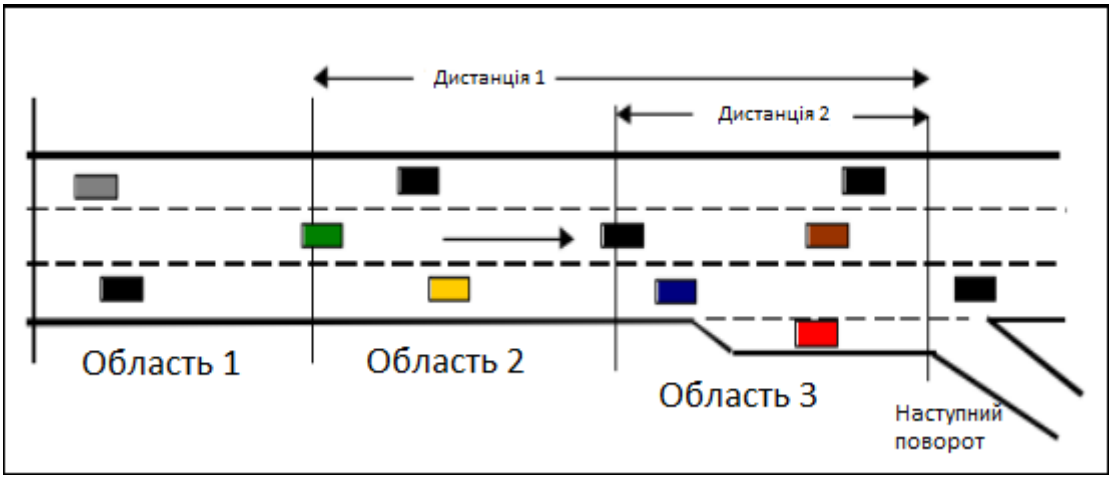


Рисунок 1.3 - Області зміни смуги

Зона 1: найбільша відстань від наступного повороту щоб виміряти як водій:

I. Змінює смугу та швидкість руху

II. Швидкість транспортних засобів перед ним та відстань

III. Швидкість майбутнього провідного транспорту на інших смугах.

Зона 2: проміжна зона. Машини шукають проміжок щоб повернути на наступну смугу що впливає на рішення про зміну смуги руху.

Зона 3: найкоротша відстань до наступної точки повороту. Автомобіль почав зміну смуги з прилеглої до наступного повороту. Під час цього необхідно зменшити швидкість щоб забезпечити достатній для іншого транспорту проміжок

Модель прийому проміжку

Прийом проміжку використовується для визначення кількості транспортних засобів у трафіку, які можуть проїхати через точку конфлікту. Проміжок визначається як час існування відстані між лідируючим та відстаючим засобом на цільовій смузі руху (Рисунок 1.4).

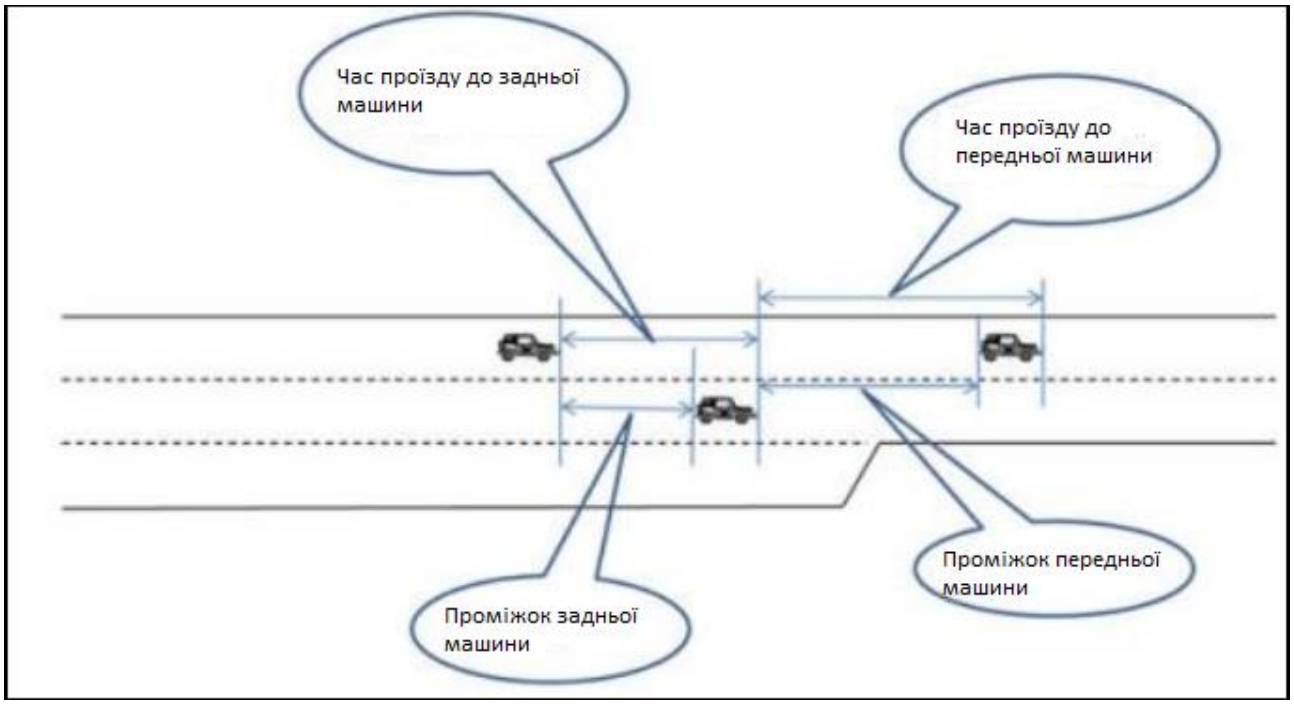


Рисунок 1.4 - Зміна ліній руху з урахуванням проміжків

Моделі прийняття проміжків використовуються для визначення розмірів проміжку який буде обрано водієм, який збирається перестроїтися або проїхати перехрестя. Критичний проміжок визначається числом прийнятих проміжків, коротших за рівне числу відхилених. Параметри, які використовує модель – це прискорення, бажана та фактична швидкість. Серед цих параметрів, найбільш

важливими є швидкість розгону, максимальний час, за який водій поступається дорогою та видимість дороги на перехресті.

Швидкість прискорення - це здатність транспортного засобу розганятися з необхідним безпечним інтервалом. Максимальний час уступки визначає, коли водій починає ставати нетерплячим, якщо він не може визначити проміжок.

1.2.2 Макроскопічний метод моделювання

Макроскопічний метод моделювання описує перетини на низькому рівні деталізації. У макроскопічній моделі, поток руху представлено сукупністю таких характеристик, як швидкість, потік та щільність. Дослідники вивчали відносини між цими характеристиками та прийшли до спроби розвинути математичний опис для кривих, які стали основою для моделей Гріншильда та Грінберга. Перша модель використовується для розробки моделі безперебійного транспортного потоку. Це відносно точний і простий метод моделювання, який використовує три характеристики у графі (Рисунок 1.5).

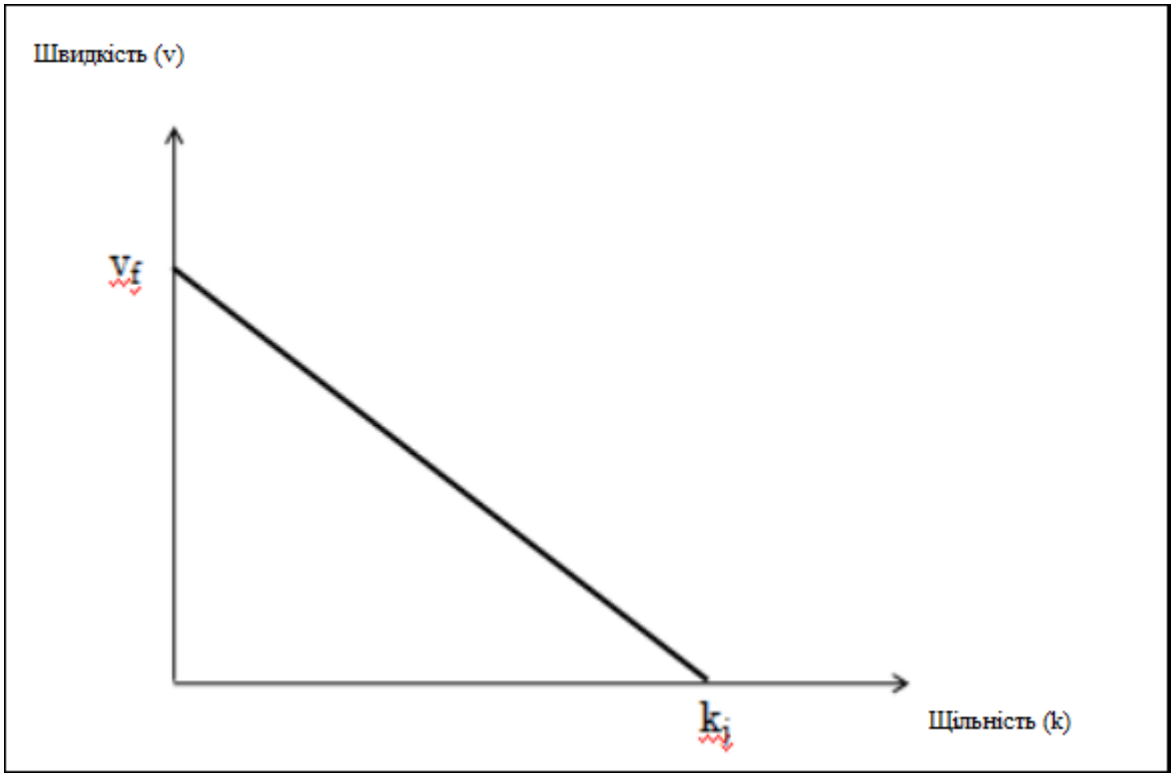


Рисунок 1.5 - Відношення швидкості та щільності

Модель Гріншильда показує, що відносини швидкості та щільності є лінійними. Зі зменшенням щільності, потік збільшується до оптимального коли на дорозі знаходиться більше машин. Швидкість також зменшується через взаємодію транспорту.

У математичному вигляді (Рисунок 1.6), характеристики моделі виглядають так:

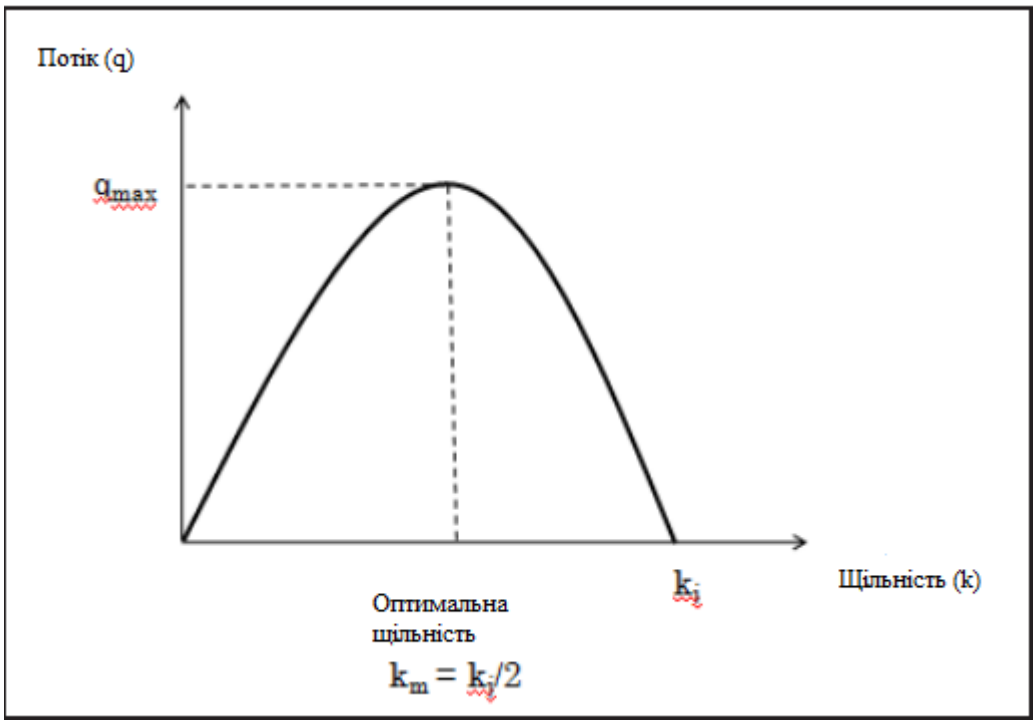


Рисунок 1.6 - Відношення потоку та щільності

I. Швидкість, v

v = vf – (vf / kj)*k, де,

v = швидкість (км на год),

k = щільність (транспорту на км),

vf = швидкість на вільній дорозі (транспорту на км),

kj = щільність затору (транспорту на км).

II. Потік, q

q = (vf – (vf/kj)*k)*k, де,

q = потік,

v = швидкість (км на год),

k = щільність (транспорту на км),

vf = швидкість на вільній дорозі (транспорту на км),

kj = щільність затору (транспорту на км).

Модель Гріншильда показує зв'язок між потоком та щільністю через параболу (Рисунок 1.7). Коли потік дуже низький, швидкість вища. Водії здатні рухатися з бажаною швидкістю. Зі збільшенням потоку, швидкість поступово зменшується. Найбільші показники потоку показують перехід від неперенавантаженого стану в перенавантажений.

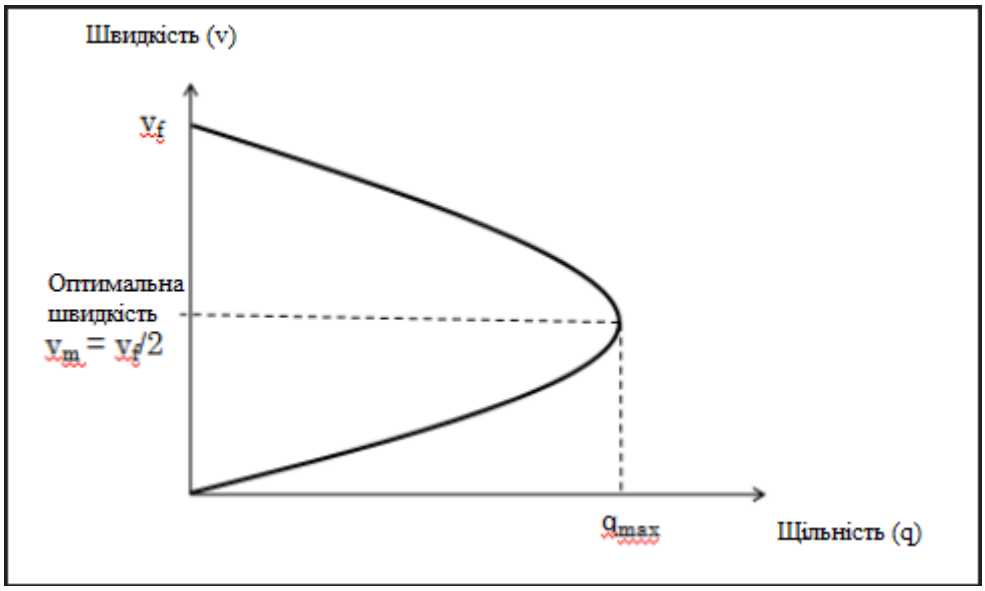


Рисунок 1.7 - Відношення швидкості (v) та щільності (k)

III. Максимальний потік

qm = vm*km

або

qm = (vf*kf)/4, де

q(max) = максимальний потік,

v(max) = оптимальна швидкість,

k(max) = оптимальна щільність.

Коли пропускна здатність дороги (qmax) доходить до максимальних значень потоку, зростає щільність та спадає швидкість через максимальну кількість транспортних засобів які пройшли через певну точку. Проміжків, які могли би

занйняти машини, немає. Характеристики перенавантаженого стабільного потоку це низька щільність та висока швидкість, коли є проміжки у прилеглій смузі.

1.2.3 Мезоскопічний метод моделювання

Мезоскопічний (проміжний) метод описує аналіз елементів транспортування невеликими групами. Мезоскопічна модель це комбінація мікроскопічної та макроскопічної моделей. У цій моделі симулюється розподіл колон. Моделі розподілу колон ураховують розподіл транспортного потоку по мірі його руху від верхнього потоку до нижнього. Вони оцінюють об'єм транспортного потоку на нижній частині на основі профілю відправлення транспортних засобів в верхній частині та середнього часу подорожі по зв'язку (Рисунок 1.8).

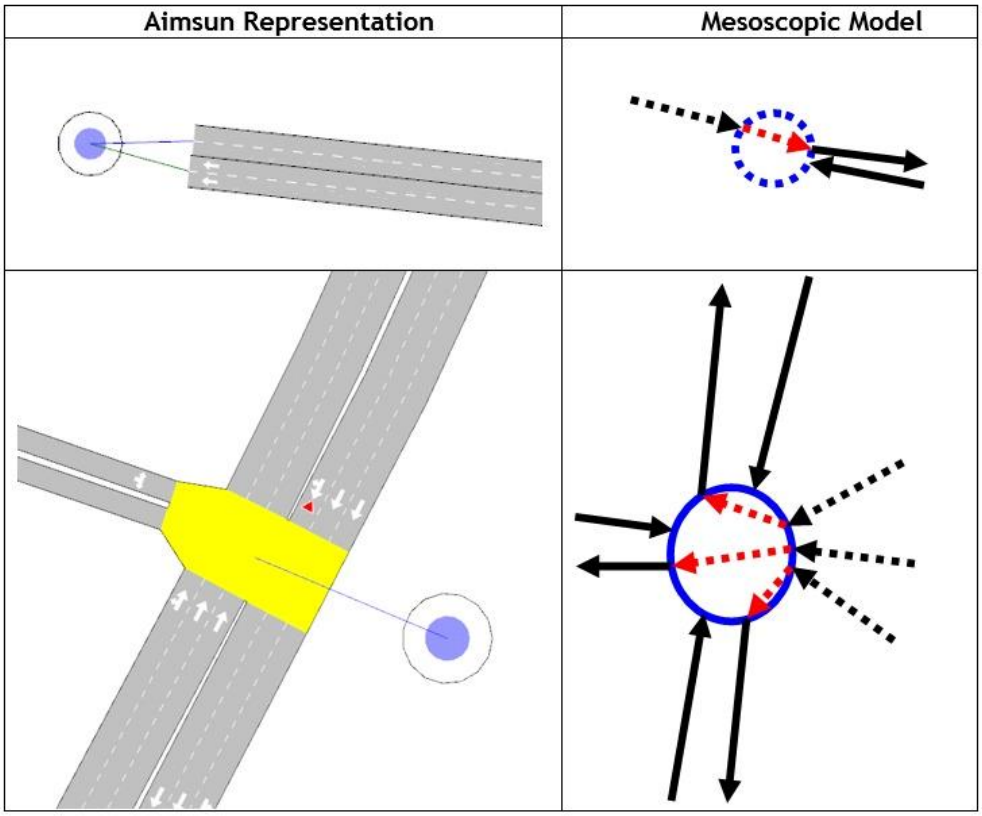


Рисунок 1.8 - Схема роботи мезоскопїчної моделі на платформі Aimsun

Існують два методи мезоскопічного моделювання: розподіл колон та поведінка транспорту у колоні.

Перший метод - це розподіл колон. Під час того, як колона рухається вниз по потоку від верхнього перехрестя, відстань між транспортними засобами може збільшуватися через різницю у швидкостях транспортних засобів, взаємодію транспортних засобів (зміна смуги та інші перешкоди), (паркування, пішоходи та інші). Цей явище називається розподілом колон.

Другий метод - це поведінка транспортних колон. Група транспортних засобів рухається на невеликому інтервалі та з однаковою швидкістю. Поведінка транспортних колон передбачає прибуття транспортних колон з часом та загальний час прибуття.

У моделюванні поведінки колон важливим врахуванням є кількість транспортних засобів-учасників колони, її швидкість та розподіл швидкості між учасниками. Колони формуються, коли світлофор переходить на червоне, і транспортні засоби починають утворювати чергу. Для одного світлофора довжина черги є ключовим показником ефективності, який впливає на розподіл прибуття транспортних засобів. Колони розсіюються, коли світлофор починає переходити з жовтого на зелений, вона починають набирати швидкість, а потім транспортні засоби починають рухатися. Як тільки транспорт виходить за лінію зупинки, він починає рухатися в стислій позиції колони з короткими інтервалами часу та розсіюється, рухаючись далі по потоку.

У таблиці 1.1 нижче представлені різні програми та моделі, які використовувалися у послідженнях з симуляції дорожнього руху. З нього можна прийти до висновку, що симуляційні моделі можуть вирішувати широкий спектр проблем, пов’язаних з питаннями транспорту та логістики.

Таблиця 1.1 Приклади програм моделювання

Тип моделі	**Назва ПЗ**	**Об'єкт дослідження**	**Мета**	**Враховані параметри**	**Ким розроблена**
Мікроскопічна/ мезоскопічна	Aimsun	Нова Зеландія, північ Шор-Сіті, Тахарото-роуд – запропонований проект швидкісного автобусного транзиту	Аналіз руху автомобілей, сигналів світлофора та час у дорозі	1) Потік 2) Швидкість 3) Час у дорозі	PTV System Software and Consulting GMBH, Франція
Мікроскопічна	SUMO	Моделювання руху на основі щоденної діяльності населення міста Кельн	Порівняння та оцінка різних алгоритмів пов'язаних з рухом, та оцінка системи світлофорів.	1) Час сигналу світлофора 2) Напрямок повороту 3) Тип перехрестя 4) Суміжні лінії	D.Krajzwwicsz, Institut Fur Verkehrssystem Technik, Берлін, Німеччина
Мікросопічна	Fresim	Симуляція транспортного потоку на магістралях у Сінгапурі	Вдосконалення геометрії та оперативних можливостей, симуляція геометричних умов	1) Вимір схилів 2) Поведінка водія 3) Знаки 4) Виїзд по схилу	Federal Highway Administration, 1996

Продовження Таблиці 1.1 Приклади програм моделювання

Мікроскопічна	Corsim	Запропоновані альтернативні дизайни Сієтла	Симуляція трафіку та контролю за трафіком	1) Знаки Стоп 2) Сигнали світлофора 3) Вимір схилів	Federal Highway Administration (FHWA), США
Макроскопічна	Visum	Планування маршрутів автобусів у Мумбаї, Індія	Аналіз потоків, прогнози та географічна інфосистема; планування послуг громадського транспорту	1) Картографування 2) Екологічні показники 3) Час у дорозі 4) Потік 5) Час затримки 6) Довжина черги	PTV VISUM
Макроскопічна	Saturn	Розрахунок часу сигналу світлофора в узгодженій мережі світлофорів у Великобританії, США та Австралії	Дослідження впливу вулиць з одностороннім рухом, контроль руху на дорогах для автобусів; аналіз та оцінка систем управління трафіком	1) Прийняття проміжків 2) Координація перехресть 3) Швидкість 4) Насиченість потоку	Institute of Transport Studies, University of Leeds, 1979

Перелік об’єктів дослідження:

https://www.transportationgroup.nz/papers/2001/T5_Haas.pdf

https://www.diva-portal.org/smash/get/diva2:772068/FULLTEXT01.pdf

https://www.researchgate.net/publication/224797921_The_application_of_microscopic_activity_based_travel_demand_modelling_in_large_scale_simulations

https://www.sciencedirect.com/science/article/abs/pii/S0965856402000034

https://trid.trb.org/view/576076

https://www.matec-conferences.org/articles/matecconf/ref/2018/09/matecconf_mucet2018_03006/matecconf_mucet2018_03006.html

1.3 Принципи роботи програм моделювання руху

1.3.1 Математичний принцип

У основі симуляції будь-якої системи лежить принцип її відтворення (моделювання). Математична модель передбачає уявлення поведінки істоти, яка буде змодельована у математичному форматі. У прикладі моделі, запропонованій нижче, ми розглядаємо інтерсекцію двух доріг у місті з помірним рухом. Параметри та робота моделі пояснюється у наступних підрозділах.

Загальні припущення

Представлена в роботі модель застосовується для відрізків доріг з помірним рухом поблизу перехрестя. Було зроблено багато припущень щоб спростити процес аналізу та зменшити складність існуючих моделей. Деякі з загальних припущень та специфікації можна побачити нижче.

- Концентрація моделі на загальному потоці руху на ділянках дороги поблизу перехрестя. Цей сегмент включає довжину приблизно від 100 до 150 метрів від перехрестя, тобто достатньої довжини щоб утримувати максимальний затор із 20 до 30 транспортних засобів. Цю довжину позначають літерою *L*.

- Ми припустимо, що усі транспортні засоби є чотириколісними. З однаковою довжиною щоб спростити обчислення. Довжину позначимо як *Lc*.
- На ділянці дорогі, який моделюється, засоби рухаються з постійною швидкістю. Це припущення виправдане, оскільки ця модель призначена для міських перехресть. Коли транспортний засіб наближається до перехрестя, очікується, що на відстані приблизно 100 метрів засіб почне сповільнюватись, оскільки ділянка переповнена і сигнали постійно змінюються. Тому прискорення транспорту вважається незначним. Постійну швидкість позначимо як *Vc*.
- Рух є виключно одномірним на прямих дорогах, і можливість будь-якого обгону ігнорується. Це припущення виправдане, оскільки зони біля світлофорів є зонами великої щільності трафіку, і водії в ідеалі не повинні робити обгони в таких зонах, оскільки це може призвести до заторів.
- Параметри моделі виражені у відношені до позиції *x* та часу t (Рисунок 1.9), також важливо пам’ятати що транспорт це не точковий об’єкт* . Тож, кожен раз, коли ми звертаємося до параметра моделі у позиції x_i, ми насправді маємо на увазі вікно розміром *Lc,* яке дорівнює довжині транспортного засобу та розташоване на позиції x_i. Це вікно називається «полосою». Тому вісь X вважається дискретною від точок *Lc, 2Lc, 3Lc* до *L*

* - У кінематиці (розділ механіки, який вивчає рух тіл без врахування діючої сили), точковий об’єкт це вираз, який означає предмет, розміри якого не враховуються або ігноруються. Точковий об'єкт - це маленький предмет, який рахується як точка, щоб спростити обчислення. Коли розмір предмета є дуже малим порівняно з відстанню, яку він пройшов, його можна вважати точковим об'єктом.

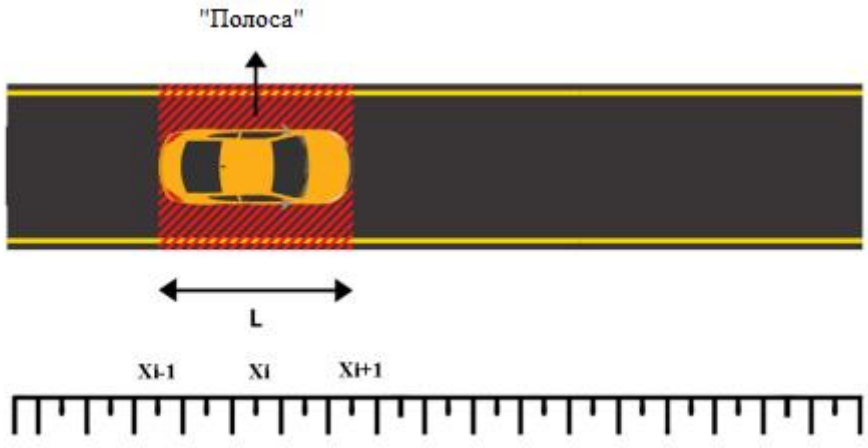


Рисунок 1.9 - Полоса» на позиції xi

Параметри моделі

Розглянемо перехрестя двох односторонніх доріг як показано на Рисунку 1.10. Потім, з деякими модифікаціями, цей аналіз може бути також розширений на багатополосні дороги.

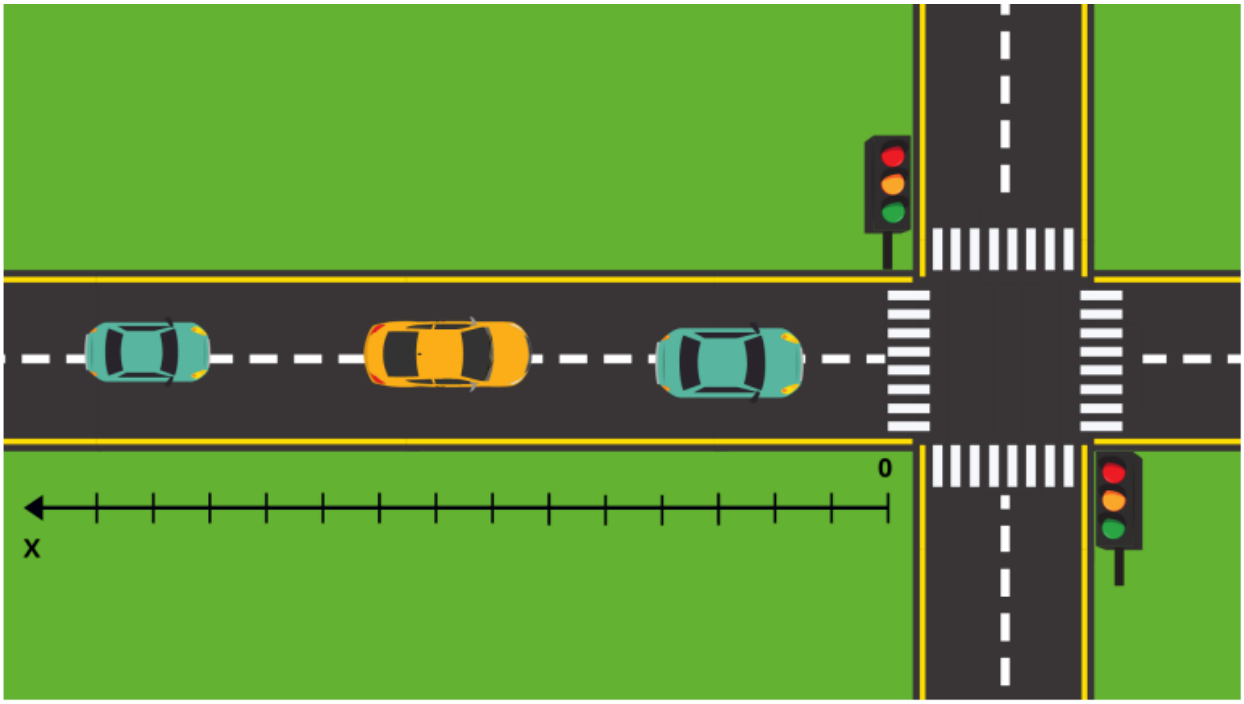


Рисунок - 1.10 Модель діаграми

Оскільки рух вважається однобічним, у нас є вісь X. Початок координат розташований у точці перетину дороги з іншою дорогою або на початку перетину. Тепер ми визначаємо деякі важливі параметри для математичного опису цієї моделі - середня щільність трафіку ρ, потік q та швидкість Vc.

Середня щільність трафіку p (Рівняння 1.1) визначається кількістю транспорту на одиницю довжини дороги. У представленій моделі, *p(x, t)* – це конкретно співвідношення кількості транспортних засобів від 0 до X за проміжок часу *t* до довжини *x*. У зонах з високими показниками заторів, особливо перед світлофорами на перехрестях доріг, щільність трафіку наближається до *pmax*. Якщо світлофор довго показує червоне світло, щільність трафіку також наближається до *pmax*. Протягом усього аналізу, ми враховували тільки середню щільність трафіку. Отже термін «щільність трафіку» у представленій роботі означає тільки середні показники щільності і визначається рівнянням 1.1. Транспортний потік у точці на дорозі (q) визначається як кількість засобів, що перетинають цю точку за одиницю часу і виражається у транспорті за хвилину або транспорті за годину часу.

$$p(x,t) = N/x \qquad (1.1)$$

N – кількість машин між 0 та X

Згідно з припущеннями, усі транспортні засоби мають однакову довжину *Lc*. Тому, максимальна можлива щільність обчислюється Потік на початковій межі дороги в момент часу *t* називається витоком $q_{in} = q(L, t)$. Це швидкість. З якою транспорт наближається до перехрестя (Рівняння 1.2).

pmax = 1/L

$$q(x,t) = N/t \qquad (1.2)$$

N – кількість транспорту

t – час

Швидкість руху потоку (*Vc*) – це швидкість, з якою транспорт рухається на конкретній вивченій дорозі. Вона виражена в кілометрах на годину (км/год) або в метрах на секунду (м/сек). Як ми вже обговорили у припущеннях, припускається,

що показник швидкості є сталим для всіх транспортних засобів, які знаходяться близько до перехрестя, та їх прискорення не враховується.

Довжина черги (*Ql)* визначається як кількість засобів, які чекають в черзі, коли світлофор загориться зеленим. *Ql* може бути порівняним з заторами або пробками. Довготривалий червоний сигнал збільшує довжину черги.

1.3.2 Модельне дослідження

Припущення та математичні параметри обговорювалися у другому підрозділі. У цьому розділі описані математичні варіації параметрів запропонованої моделі для двох важливих умов дорожнього руху: коли світлофор переходить від зеленого сигналу до червоного, та навпаки.

1) Сигнал переходить від зеленого до червоного

Припустимо, що сигнал стає червоним у момент часу $t = tr$. Коли світлофор загорається червоним, жоден транспортний засіб (з деякими вийнятками) не може проїхати перехрестя. Ми припустили, що всі учасники дорожнього руку рухаються зі сталою швидкістю *Vc*. Автомобілі продовжуватимуть рухатися з цією швидкістю, поки вони мають доступний маршрут, а потім зупиняться, що призведе до формування черг. Тут описано зміни в кількості транспорту, довжини черги та щільності трафіку в залежності від часу та положення.

1. Різниця у кількості транспорту: Виток заданий як *qin(t)*. У будь-який момент часу *t*, такий що $t>tr$, загальна кількість транспорту на дорозі в межах симуляції задається у Рівнянні 1.3

$$N(t) = \left\{ \int_{tr}^{t} q(L,t)dt \right\} + Lp(L,tr) \tag{1.3}$$

У рівнянні вище, перший член позначає кількість транспортних засобів, які з’являються сценарії між часами *tr* та *t*. Другий член позначає кількість засобів, які вже присутні у сценарії в час $t = tr$.

2. Різниця у середній щільності трафіку: Для вивченя зміни щільності трафіку *p(x, t)* як функції часу, коли сигнал стає червоним, спробуємо оцінити вираз для *p(x, t)* таким чином, що *t>tr* і *t – tr* є малим показником. У такому випадку *p(x, t)* повністю залежить від *p(x, tr,* оскільки припускається, що інтервал часу невеликий і жоден засіб, не задіяний у сценарії раніше, не увійшов у регіон *x*. Як показано на Рисунку 1.11 нижче, коли сигнал стає червоним, вихідний потік *qin* стає рівним нулю, і отже, щільність трафіку починає зменшуватись до *x=0*.

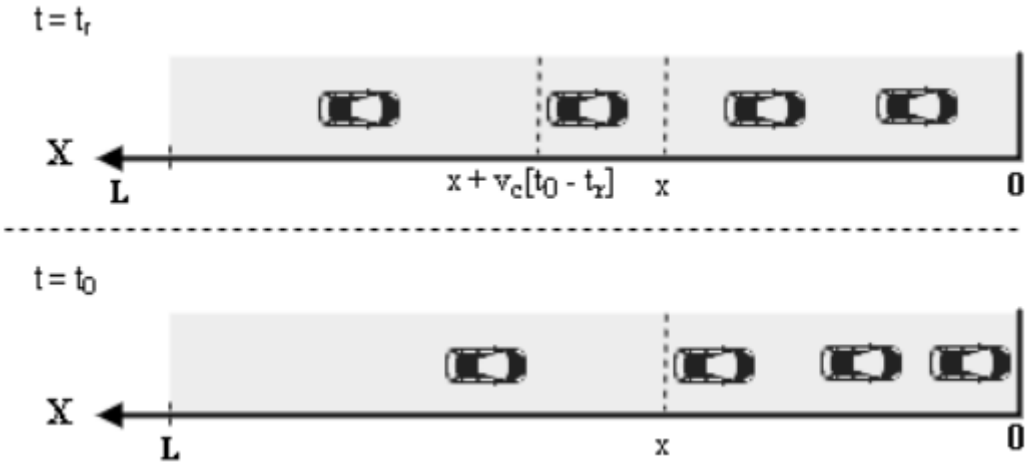


Рисунок - 1.11 Зміна щільності трафіку

Можна зробити висновок, що будь-який транспортний засіб, який перебуває в регіоні між x_0 $x_0 + Vc(t_0 - t_r)$, повинен перетнути $x = x_0$ до моменту часу $t = t_0$. Оскільки світлофор горить червоним, жоден транспортний засіб не виїхав, і тому загальна кількість транспорту у регіоні $0 <= x <= x_0$ в час $t = t_0$ довівнює кількості засобів, які вже перебували в цьому регіоні в час $t = t_r$ плюс новий транспорт, який увійшов в регіон, що ефективно дорівнює загальній кількості транспортних засобів у регіоні $0 <= x <= x_0 + Vc(t_0 - t_r)$ в час $t = t_r$.

Число машин ($0 <= x <= x_0$ при $t = t_0$) = Число машин ($0 <= x <= x_0 + Vc[t_0 - t_r]$ при $t = t_r$)

Тому, щільність трафіку може бути визначена у загальному вигляді, як показано у Рівнянні 1.4 нижче.

$$p(x,t) = p(x + Vc[t - tr], tr)\{1 + (Vc[t - tr]/x\} \qquad (1.4)$$

Однак, цей вираз є вірним лише для малих значень $t-t_r$, припускаючи, що жоден новий транспортний засіб, який увійшов у сценарій, не перетнув x_0. Коли t_0 стає великим та умова $x_0+v_c[t-t_r]>L$ стає актуальною, тоді загальна кількість транспортних засобів у регіоні $0 <= x <= x_0$ включає транспорт, розглянутий у попередньому аналізі, а також засоби, які увійшли в сценарій в результаті витоку (Рівняння 1.5). Будь-який витікаючий транспорт в $x = L$ перетне $x = x_0$ після часу $t=(L- x_0)/ Vc$. Позначимо цей час як $\tau(x_0)$.

$$\tau(x_0) = (L - x_0)/ Vc \qquad (1.5)$$

Очевидно, що у момент часу $t = t_0$, такому що $x_0 + Vc[t-t_r]>L$, щільність трафіку $p(x_0, t_0)$ задається у наступному виразі. Перший член враховує кількість транспортних засобів, які вже присутні на дорозі в момент часу $t = t_r$, а другий член враховує нові транспорти, які виїхали на дорогу під час витоку після моменту часу $t = t_r$, і перетнули $x = x_0$ (Рівняння 1.6)

$$: \rho(x0, t0) = 1/ x0 \left\{ p(L, tr).L + \int_{t=tr}^{tr+r(x0)} q(L,t)dt \right\} \qquad (1.6)$$

Ще один важливий аспект, який слід враховувати, це те, що $p(x, t)$ обмежена та не може бути більшою, ніж 1/L для дороги з однією смугою руху. Зокрема, для дороги з n кількістю смуг, максимальне значення $p(x, t)$ є n/L. Кінцевий вираз для $p(x, t)$, використовуючи усі розглянуті вище приклади, наведений у Рівнянні 1.7.

$$\theta 1(x,t) = p(x + Vc[t - tr], tr \{1 + (Vc[t - tr])/x\} \qquad (1.7)$$

3. Різниця у довжині черг: Кожного разу, коли світлофор переходить з зеленого на червоний, черга починає утворюватися. Внаслідок цього, довжина черги Ql починає збільшуватись. Транспорт на будь-якій позиції x входить у чергу після певного проміжку часу, якщо продовжує горіти червоне світло. Важливо

зазначити, що перед тим, як транспорт увійде у чергу, усі транспортні засоби перед ним теж повинні увійти у чергу. Ми припускаємо, що всі вони рухаються зі сталою швидкістю *Vc* і потім зупиняються після входу в чергу. Припускається також, що дорога має тільки одну полосу. У випадку багатополосних доріг, транспортні засобу будуть розподілюватись між кількома полосами, щоб зменшити ефективну довжину черги *Ql* як функції часу (Рисунок 1.12). Для будь-якого часу $t = t_0$ такого, що $t > t_r$, засіб у черзі повинен пройти максимальну відстань $Vc\ (t_0 - t_r)$.

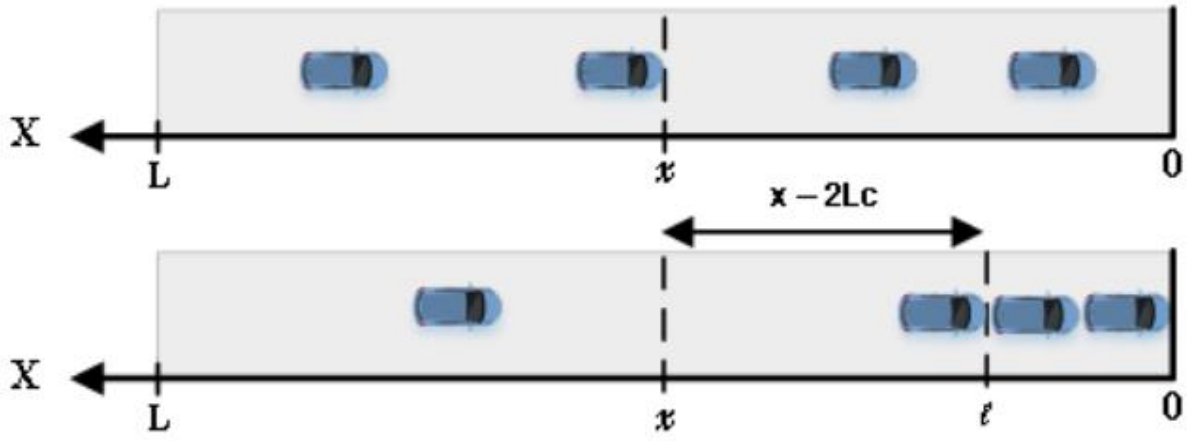


Рисунок 1.12 - Утворення черги

Іншими словами, можна зрозуміти, що для транспорту, щоб бути частиною черги в момент часу t_0, його положення x_i в час $t = t_r$ повинно задоволняти умову $x_i <= Vc*(t_0 - t_r)$. Отже, ми приходимо до висновку, що

*Ql(*t0*) ∞ (Кількість транспорту x<= Vc(t0 – tr) у t = tr)*

Ql(t0) ∞ *p(Vc[t0 – tr] у t = tr)*

Довжина кожного транспортного засобу вважається сталою *Lc*. Отже, ефективна довжина черги в час $t > tr$ задається Рівнянням 1.8:

$$Ql(t) = p(Vc[t - tr], tr)Vc[t - tr]Lc \qquad (1.8)$$

У випадку багатополосних доріг з n полосами, кожна полоса формує окрему чергу, тому середня довжина черги приблизно в n разів менша, ніж довжина черги в у випадку доріг з однією полосою. Таким чином, ми можемо узагальнити цей вираз як Рівняння 1.9

$$Ql(t) = Lc/n\{p(Vc[t - tr], tr)Vc[t - tr]\} \quad (1.9)$$

Зазначений вище вираз актуальний лише для засобів, які знаходяться у межах моделі у час $t = tr$. Коли світлофор згорається червоним, потік на x = 0 стає рівним 0, тобто витікаючий потік стає нульовим, але це не впливає на виток. Виток *qin* – це потік на x = L. Отже, нові засоби додаються до нашої моделі та до черги через певний час. Якщо її довжина Ql(t) перевищує межу моделювання, настає ексремальний затор. Ткі ситуації не розглядаються у запропонований моделі, оскільки вона розроблена лише для аналізу помірного трафіку.

2) Сигнал переходить від червоного до зеленого

Припустимо, що світлофор загорається зеленим в момент часу $t = tg$. Коли

це відбувається, транспорт може перетинкати перехрестя, і тому довжина черги починає зменшуватися. Ми припусаємо стан транспортного руху під час $t = tg$ як початковий стан і весь аналіз представлений у термінах початкових умов. Довжина черги Ql(tg), щільність трафіку *p(x, tg)* та кількість засобів N(tg) задаються схожим чином.

Різниця у кількості транспорту: у будь-який момент часу $t = tg$, такий що $t0 > tg$, кількість засобів всередині меж моделі залежить від кількості засобів, які вже перебувають в межах моделі в час ($t = tg$), кількості засобів що вже перетнули перехрестя між $t = tg$ та $t = t0$, та кількості нових засобів що вже увійшли в межі моделі протягом цього інтервалу часу (Рівняння 1.10).

$$N(t) = N(tg) - \frac{Vc}{Lc} * [t - tg] + \int_{tg}^{t} q(L, t)dr \quad (1.10)$$

У рівнянні 1.10, ми припускаємо, що час t достатньо малий, щоб уся черга, яка була створена перед $t = tg$, ще не перетнула перехрестя. Час tg, коли уся черга покидає перехрестя задається Рівнянням 1.11:

$$tg = tg + (\frac{Ql(tg)Lc}{Vc}) \tag{1.11}$$

Отже, рівняння 1.11 вірне лише для tg <t< tg. Для $t > tg$ уся черга очищається, тобто затор розсіюється і досягається рівномірність у потоку. Тепер кількість засобів в сценарії залежить лише від швидкості притоку та витоку. Отже загальна кількість засобів задається Рівнянням 1.12:

$$N(t) = \int_{tg}^{t} q(L, \tau) - q(0, \tau)\, d\, \tau \tag{1.12}$$

Перший термін позначає кількість засобів, що входять у сценарій, а другий позначає кількість засобів, що його залишають. Оскільки припускається, що швидкість транспорту є постійною, коли світлофор горить зеленим, потік руху однорідно розповсюджується. Іншими словами, можна сказати, що потік на позиції x = L в момент часу t = τ розповсюджується до позиції x = 0 в момент часу t = τ + (L/Vc). Отже, ми маємо таке Рівняння 1.13:

$$q(0, t) = q(L, t - L/Vc) \tag{1.13}$$

Використовуючи усі аналізи та обговорення подані вище, загальна кількість транспортних засобів на дорозі може бути задана узагальненим Рівнянням 1.14:

$$\theta 1(t) = N(tg) - \frac{Vc}{Lc} * [t - tg] = \int_{tg}^{t} q(L, t) dr \tag{1.14}$$

$$\theta 2(t) = N(t) = \int_{t}^{tg} [q(L,\tau) - q\left(L, \tau - \left(\frac{L}{Vc}\right)\right] d\tau$$

$$N(t) = \begin{cases} \theta 1(t) \text{якщо } tg < 5 \leq tg \\ \theta 2(t) \text{якщо } t > tg \end{cases}$$

Різниця середньої щільності також може бути отримана за допомогою подібного аналізу. Вирази для різниці середньої щільності не представлені у роботі, оскільки вони виходять за межу області інтересів. Як тільки світлофор загорається зеленим і затор розсіюється, отримується бажаний рівномірний потік трафіку, і єдині параметри, які цікавлять нас у такій сиутації це швидкість притоку та витоку. Однак ці вирази також можуть бути оцінені та використані для специфічних результатів.

1.4 Об'єктно-орієнтоване моделювання трафіку за допомогою Java

У наведеній роботі, для розробки симуляції використовується мова програмування Java. Це високорівнева об’єктно-орієнтована мова, яка підтримує багатопотокові операції, що робить її популярним вибором при розробці ПЗ. Набір розробки Java JDK постачається з багатьма корисними вбудованими інструментами та пакетами, які допомагають у процесі розробки. Завдяки цим особливостям запропонована робота використовує Java як мову програмування. Однак процес моделювання та розробки є загальним і може бути реалізованим за допомогою будь-якої відповідної мови (себто C++, GO, Python) та для будь-якої платформи. Для побудови інтерактивного графічного інтерфейсу (GUI) використовується фреймворк JavaFX.

1) Архітектура симуляції

Архітектура запропонованого дизайну симуляції виконана у вигляді кількох класів Java. Клас Simulation є верхнім виконавчим класом, в якому знаходяться усі інші компоненти. Файл сценарію містить повну інформацію про сценарій, який симулюється разом з усіма об’єктами задіяними у сценарії. Параметри

математичних моделей надаються класом Scenario Handler. Об’єкти симулюються послідовно в циклічному порядку протягом певного часу класом Controller. Детальний опис різних класів у дизайні обговорюється в цьому підрозділі та на Рисунку 1.13.

Інформація про реальний сегмент, який представлено у симуляції, знаходиться у файлі Scenario. Сценарій містить колекцію різних об’єктів, які є частиною сценарію. Ці об’єкти можуть представляти фізичні об’єкти реального життя або віртуальні системи підтримки для керування реальними об’єктами. Наприклад, світлофор це реальний об’єкт, а контролер світлофора представляє собою логіку контролю за світлофором. Об’єкти у запропонованому дизайні симулятора можуть реалізовувати будь-який з наступних інтерфейсів.

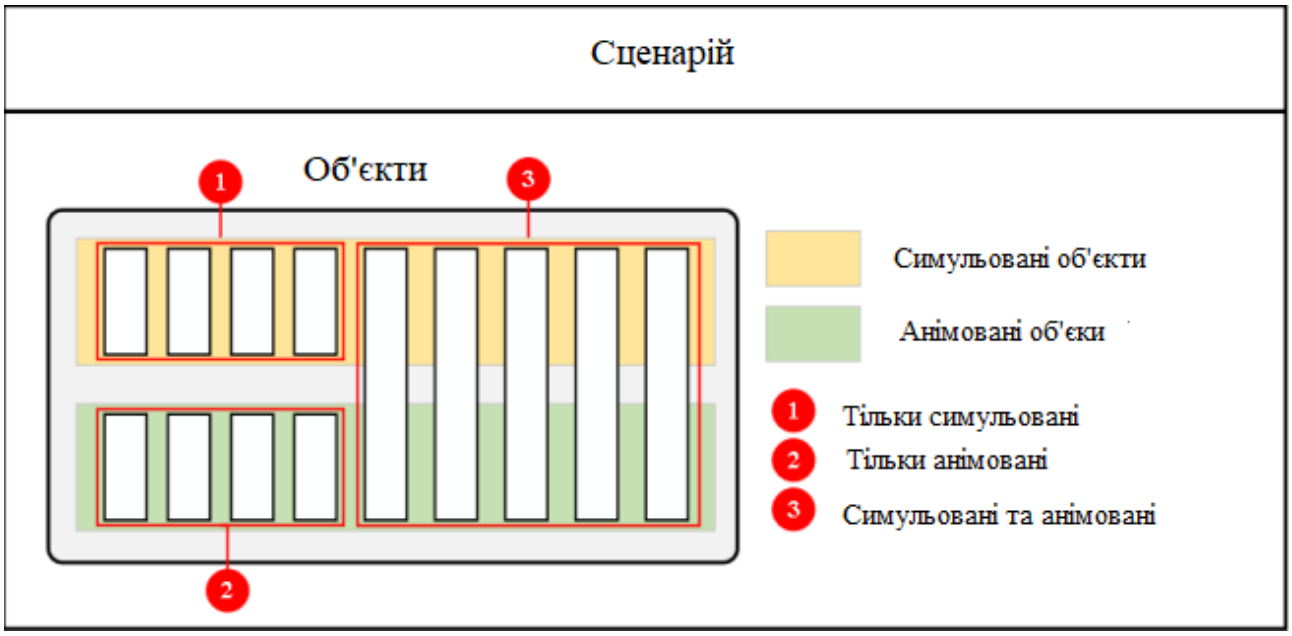


Рисунок - 1.13. Сценарій

I. Анімований Інтерфейс: Об’єкти, що формують цей інтерфейс, є частиною інтерактивного графічного інтерфейсу симулятора. Ці об’єкти мають параметри висоти, ширини, орієнтації, графічного зображення, позиції по *x* та *y*. Усі об’єкти, що представляють собою фізичні об’єкти з реального життя в сценарії, реалізують Анімований Інтерфейс. Таким чином, транспортні засоби, світлофори та дороги в сценарії формують цей інтерфейс. Всі об’єкти цього інтерфейсу повинні оголосити

метод drawOnGUI(), який дозволяє анімувати ці об’єкти на екрані графічного інтерфейсу в кінці кожного циклу симуляції.

II. Симульований Інтерфейс: об’єкти, стан яких змінюється впродовж сценарію, формують Симульований Інтерфейс. Усі динамічні об’єкти, які зазнають певних змін протягом часу, реалізують цей інтерфейс. Ці об’єкти симулюються контролером симуляції протягом кожного її циклу. Вони повинні оголосити метод simulate(), який дозволяє їх симулювати в кожному циклі.

Клас Simulation Controller відповідає за запуск, контроль та завершення симуляції. У цьому класі є список усіх симульованих об’єктів в сценарії протягом кожного циклу. Після цього він оновлює графічний інтерфейс в кінці циклу, перемалювавши усі анімовані об’єкти. Контролер також відповідає за звільнення пам’яті шляхом видалення об’єктів, які вийшли за межі сценарію. Це дозволяє сміттєзбиральнику JVM (Garbage Collector) звільнити деякий простір у купі пам’яті (heap memory), видаливши непотрібні об’єкти. Під час цього, контролер додає нові об’єкти до сценарію, такі як транспорт в нашому випадку, згідно з рівнем вхідного потоку, налаштованим у системі. Наприклад, у сценарії показаному на Рисунку 1.14, машина А щойно увійшла у сценарій. Це означає, що машина А була додана до сценарію контролером в кінці попереднього циклу симуляції. Так само, машина Б на сценарії збирається вийти за межі в кінці поточного циклу. Тому контролер видалить об’єкт, що відповідає цій машині щоб звільнити пам'ять.

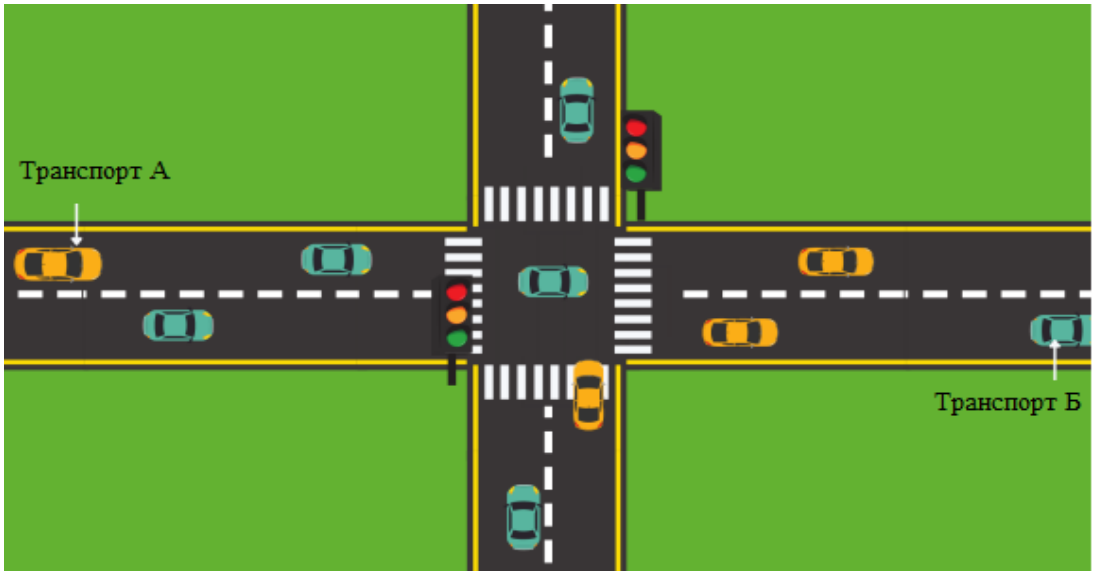


Рисунок - 1.14 Приклад сценарію симуляції

Контролер симуляції також відповідає за завершення симуляції у випадку будь-якої помилки . Таким чином, клас Simulation Controller є «серцем» запропонованого дизайну симулятора. Його функціонування описано на Рисунку 1.15.

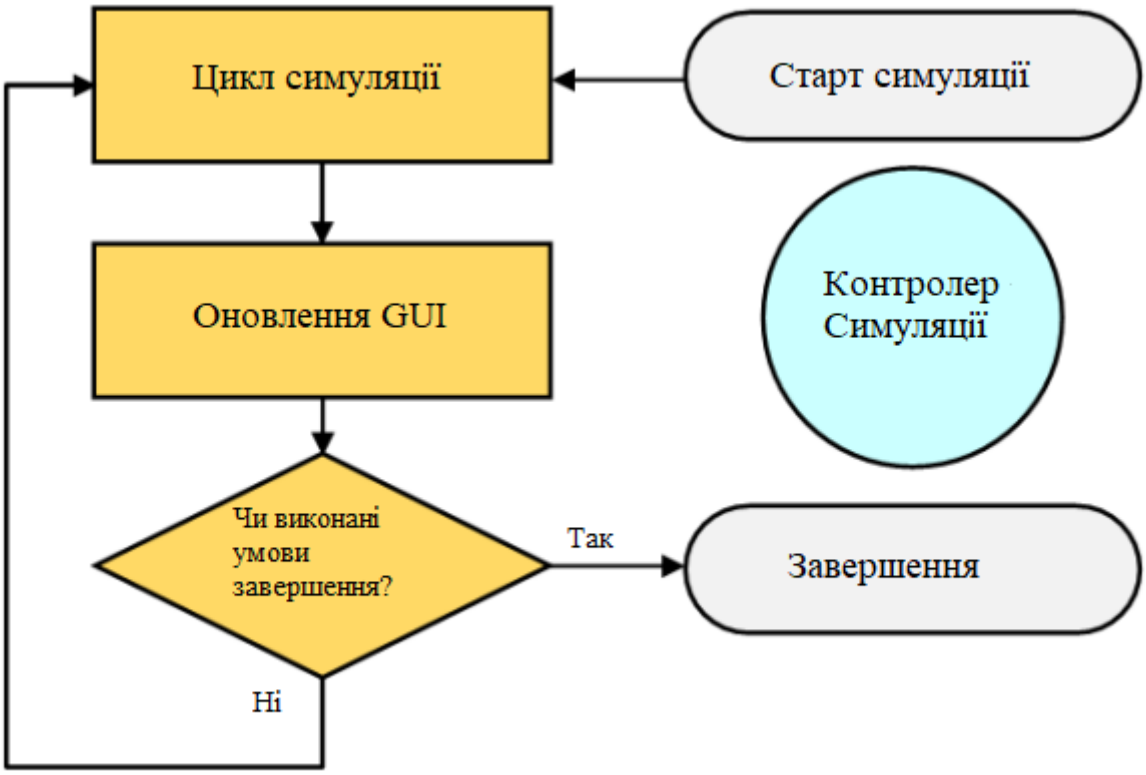


Рисунок 1.15 - Принцип роботи контролеру

Цикл симуляції – це один пройдений цикл, під час якого усі об’єкти у сценарії симулюються один за одним протягом одиниці часу симуляції. Черга всіх об’єктів, що підлягають симуляції, зберігаються в класі контролера симуляції, і всі об’єкти в черзі виконують метод simulate() один за одним. Алгоритм циклу показаний на Рисунку 1.16.

2) Керування часом симулятора

Будь-яка симуляція проводиться з урахуванням параметра t, тобто часу. Протягом кожного циклу, об’єкт симулюється протягом певного інтервалу часу. Цей інтервал часу може змінюватись і називається Часом Роботи Симуляції (Ts).

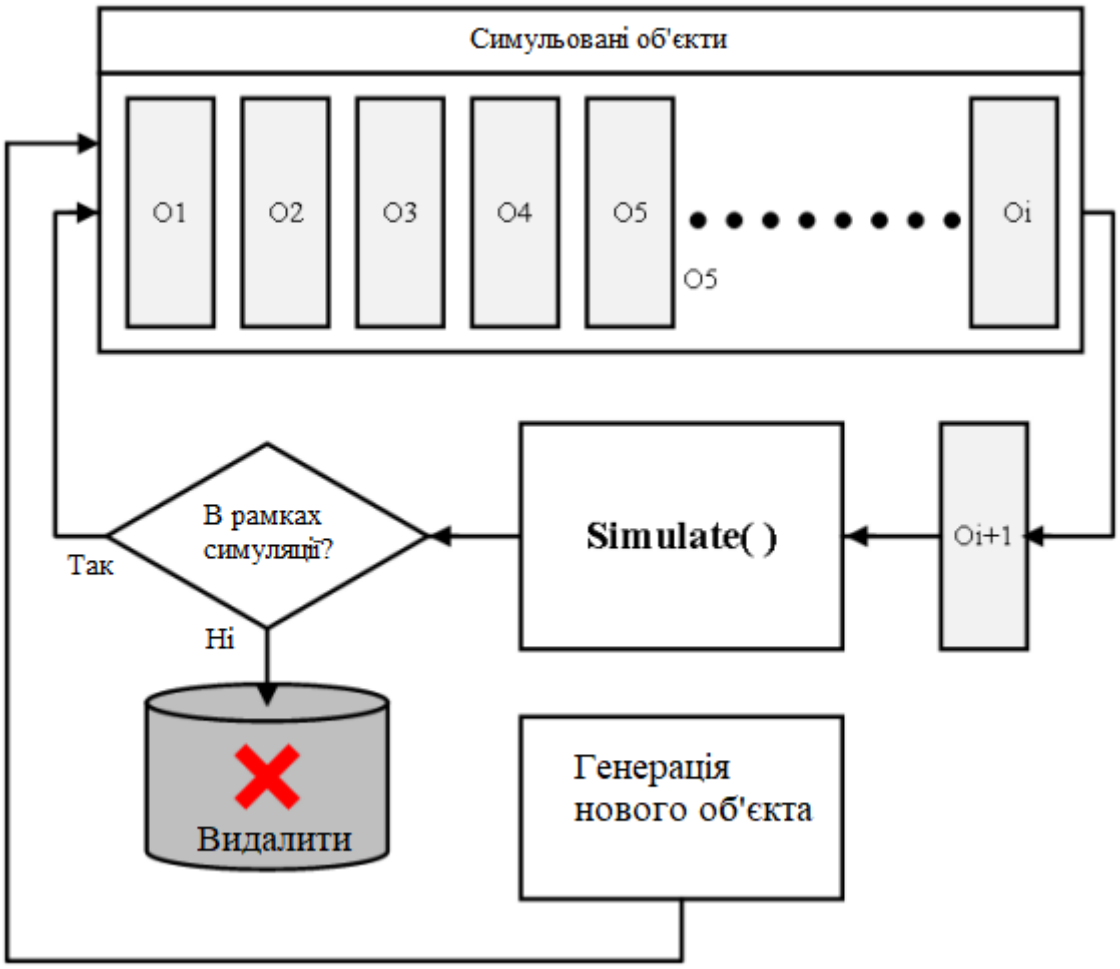


Рисунок 1.16 - Цикл симуляції

Розглянемо автомобіль A на позиції x, який симулюється протягом циклу симуляції. Швидкість на позиції x у час t задається як *v(x, t)*. Таким чином, симуляція автомобіля A в цьому циклі призведе до того, що він переміститься на відстань еквівалентну *v(x, t)*Ts*. Симуляція – це віртуальне відображення сценарію реального світу. Події реального світу масштабуються та виражаються з урахуванням реального часу. Так само, в віртуальному відображенні події виражаються та масштабуються з урахуванням віртуального часу. Ця взаємодія між реальним і віртуальним часом може змінюватись та, в основному, контролює швидкість симуляції.

Цикл симуляції спрацьовує завдяки слухачем подій tick, реалізованому через об’єкт таймера. Таймер генерує подію tick з інтервалом кожних одиниць Tr. Цей інтервал встановлюється програмістом. Він повинен бути достатньо великим, щоб

процесор завершив виконання циклу перед тим, як відбудеться наступний tick. Протягом Tr одиниць часу у реальному світі пройшло Ts одиниць часу у віртуальному. Тут ми визначаємо термін, відомий як коефіцієнт швидкості, який обчислюється за виразом нижче:

Рівняння 15: Фактор Швидкості = Tr/Ts

Кількість Tr визначає, як часто процесор виконує кроки симуляції, а Ts – скільки часу об'єкт симулюється протягом одного дискретного циклу симуляції.

3) Контролер параметрів моделі симулятора

Модельні параметри, обговорені у попередніх розділах, встановлюються та керуються в класі Scenario Handler. Цей клас, який визначається користувачем, контролює параметри моделі в різні моменти часу. У запопронованій моделі, потік на x=L та швидкість транспортного засобу, *vc*, є двома зовнішними незалежними параметрами, які повинен визначити користувач, що контролює симуляцію для отримання бажаних результатів. Решта всіх ігших параметрів є залежними та автоматично керуються відповідно до змін у незалежних параметрах. Ці параметри можуть бути встановлені як на постійне значення, так і можуть змінюватись відповідно до задачі. Такі опції налаштування забезпечують високий рівень гнучкості для проведення симуляційних тестів для різних ситуацій.

Логіка контролю сигналів визначається в класі Signal Controller. Цей клас має доступ на читання до всіх інших параметрів сценарію для прийняття рішень. Цей клас повністю визначається користувачем, оскільки користувач визначає логіку контролю світлофорів. Він може бути встановлений на фіксований час роботи або на деякий алгоритм прийняття рішень на основі змін у параметрах моделі.

Результати

З використанням архітектури та дизайну програми-симулятора, запропонованих у підрозділі 1.4, був розроблений симулятор. Цей симулятор потім використовувався для тестування простих сценаріїв дорожнього руху для перехрестя двох односторонніх доріг з однією та двома смугами. На Рисунку 1.17

показано приклад графічного інтерфейсу користувача симулятора. Зміни в реальному часу в щільності трафіку, довжині черг та кількості транспортних засобів спостерігалися на анімованому графічному інтерфейсі, розробленому за допомогою JavaFX, що відображав зміни в системі на задньому плані у вигляді анімації. На завершення симуляції, цю саму інформацію відображено у графічному форматі для кращого аналізу.

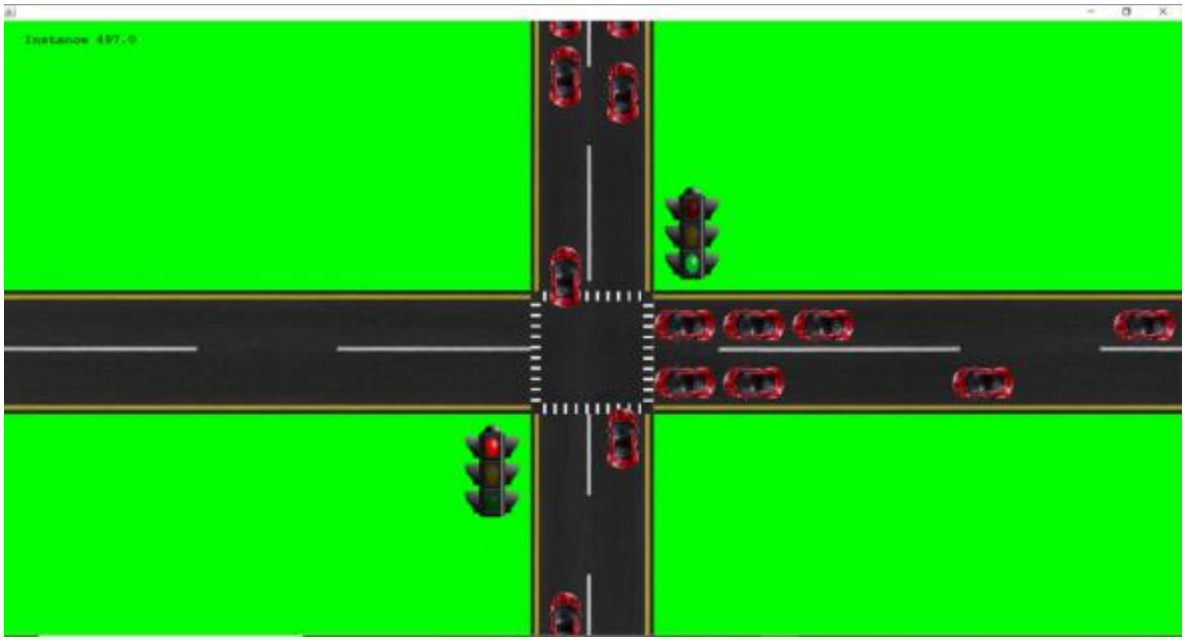


Рисунок 1.17 - Вікно роботи симуляції

Тести симуляції були проведені на перехресті двох односторонніх доріг з різними значеннями витоків. Після цього результати у форму анімації та графіків були перегланути та проаналізовані. У таблиці 1.2 нижче були наведені характеристики проведених тестів:

Таблиця 1.2 Характеристики проведених тестів

Параметр	Показники
Час припинення роботи	500
Час роботи	0.15 сек.
Фактор швидкості	3
Інтервал подій (Tr)	50 мілісек.
Частота симуляції (Fr)	20 циклів в секунду
Швидкість транспорту (Vc)	40 км/год
Довжина транспорту (Lc)	4 м.

1.5 Особливості та тренди моделювання руху

У цьому розділі розглядаються тренди та спільні характеристики між різним ПЗ для моделювання руху на дорогах та їх результати на основі дослідження Проекту «Intelligent Transport Systems Austria West», який був замовлений Управлінням провінційного уряду Верхньої Австрії. Метою проекту було реалізувати та впровадити програмний інструмент моделювання для надання реального часу оцінки дорожнього руху та короткострокових прогнозів для менш пріоритетних доріг Верхньої Австрії. Ця мережа включає приблизно 6000 кілометрів гетерогенних доріг: вуличні вулиці в містах та містечках, а також сільські дороги з гострими / сліпими поворотами, заторами, підйомами та пішохідними переходами або дороги з двома або більше смугами без перехрестів, вплив на дорожній рух яких схожий на автостраду. Крім того, потрібно враховувати гетерогенний рух: автомобілі, вантажівки, автобуси, сільськогосподарські трактори, мотоцикли, велосипеди та пішоходи.

Порівнюючи оновлені результати з ранішніми дослідженнями, здається, що деякі системи симуляцій були розроблені швидше за інші. Це також розкриває подальший розвиток деяких продуктів для їх адаптації до нових областей застосування. У зв’язку з тим, що традиційні програми не були розроблені для використання в цій області, цей крок, здається, є необхідним.

1) Системи симуляції (TSS) використовуються різними групами користувачів і повинні надавати багато різних функцій.

Існують різні групи користувачів, такі як дослідники, консультанти, оператори високо пріоритетних доріг або міські та дорожні влади. Оскільки їм потрібно вирішувати різні проблеми, вони потребують різний функціонал продукту. Для міського планування та розробки планів управління світлофорами офлайн-симуляція є достатньою. Для прогнозу часу подорожі та управління подіями в центрах управління рухом онлайн-симуляція є критичною. Багато користувачів шукають мікроскопічні деталі на макроскопічному рівні. Це можна

тлумачити як їх потребу в TSS, яке потребує менше введення даних і калібрування для надання реалістичних результатів на детальних рівнях. З іншого боку, розробники TSS намагаються надати продукт, який може працювати з різними галузями застосування. Тому вони постачають інструменти моделювання, які пропонують зростаючий функціонал. Більшість користувачів не можуть впоратися з таким функціоналом.

Прогнозується, що в майбутньому збільшиться кількість міських та дорожніх влад, які будуть використовувати ці системи частіше. З урахуванням поточних тенденцій, до додаткових груп користувачів TSS віднесуться постачальники громадського транспорту. Крім того, всі ці групи користувачів будуть потребувати більшої функціональності для розв'язання подальших завдань. Міські та дорожні влади будуть використовувати TSS не тільки для управління дорожнім рухом, а й для оцінки споживання енергії та викидів внаслідок загальної тенденції оцінки впливу дорожнього руху на навколишнє середовище та оптимізації споживання енергії. Тому вимоги до TSS будуть зростати.

2) Використання TSS для ATMS (розширені симулятори системи управління трафіком) вимагає особливих вимог до симуляторів.

Використання симуляції дорожнього руху у реальному часі є відносно новим і з ринкової точки зору все ще недооціненим. На сьогоднішній день дуже обмежене число реальних застосувань систем. Більшість цих застосувань - це наукові дослідження в академічних випадках. Існує багато виробників, але лише декілька з них постачають підходящі продукти для таких реального часу застосувань. Ці добре встановлені продукти мають стабільну спільноту користувачів. Для роботи з реальними TSS необхідні хороші моделі поведінки, алгоритми відповіді водіїв та прогнозування. Тенденція використання TSS для ATMS створила збільшений інтерес до мезоскопічних рішень, їх можливість масштабування широких територій без великої втрати точності у відтворенні динаміки дорожнього руху. Однак виробники TSS не пропонують всі ці функціональності в одному єдиному продукті. Шляхом надання інтерфейсів в TSS виробники забезпечують можливість

реалізації відсутніх функціональностей. Однак для налаштування TSS потрібно провести багато досліджень, кодування та калібрування. Тому це вимагає більше, ніж просто інструмент з точки зору користувача. Реальний час симуляції дорожнього руху потребує набору програмних інструментів, знань і, іноді, навичок. Ще одним викликом буде обробка великого обсягу даних, наданих автомобілями, мобільними телефонами або боковими датчиками. Всі вони повинні бути інтегровані в TSS для надання реального часу оцінки дорожнього руху, прогнозування та передбачуваного маршруту. Прогнози потрібні для передбачення заторів та реакції водіїв на будь-які вказівки, які надаються їм. Деякі системи реального часу без складних моделей прогнозування існують на практиці. Прогнозні методології в реальному часі все ще перебувають в дослідницькій/академічній області.

3) Для симуляції руху на різнорідних дорогах потрібна деталізована модель.

Потреба у деталізованій мережевій моделі не лише необхідна для планування, але й для симуляції в реальному часі. У обох випадках на мережі взаємодіють довгі та короткі поїздки. Крім того, у випадку виникнення випадкових подій або заторів на високопріоритетних дорогах, менш важливі дороги стають більш важливими, оскільки вони служать для переадресації транспортного потоку. Однак більшість дослідницьких завдань вже працюють з гетерогенними дорожніми мережами, і внаслідок цього TSS вже мають різні категорії доріг, інтегровані сьогодні.

4) Географічні інфосистеми (ГІС) є важливим інструментом для ефективного використання TSS.

ГІС, як підтримуюча система для ТСС, відіграють важливу роль у забезпеченні зв'язку між плануванням, прогнозуванням, операціями та управлінням, що призводить до нових інсайтів у кожній з цих областей та більш розумного прийняття рішень для підвищення ефективності. Крім того, ГІС

покращує однорідність даних. Однією з ключових функцій ГІС є надання масштабних карт для розробки географічно точних дорожніх мереж. Однак потенціал ГІС у злитті даних з різних контекстів як спільна платформа часто не визнається виробниками. Це може призвести до помилкового розвитку продукту, який відходить від сумісності та залишить користувачів в розгубленими через нерозуміння, як слід здійснити інтеграцію.

5) Симуляціїї в сільских районах стикаються з різними проблемами.

Порівнюючи сільські райони з міськими та міжміськими районами, виявляється, що сільські райони мають справу з моторизованим і немоторизованим транспортом. Оскільки на цих дорогах взаємодіють пішоходи, велосипеди, автомобілі, вантажівки, автобуси та інше, параметри вимагають калібрування. Вантажний транспорт також потрібно належним чином калібрувати, щоб врахувати ефекти вантажівок на затори. Для моделювання гетерогенного трафіку можливо змінювати параметри в мікроскопічних системах симуляції та розробляти мезоскопічні моделі. TSS повинні адаптувати свої методи розподілу на різні категорії доріг: сільські, міжміські та міські. У деяких випадках застарілі системи управління активами та мережеві інформаційні системи повинні бути замінені більш комплексними базами даних. Крім того, низька якість даних, невеликий обсяг даних у реальному часі та витрати на зв'язок у віддалених сільських районах є загальними викликами – не лише для операторів, але і для виробників.

6) TSS повинна забезпечувати ефективні системи для налаштування моделей.

Для моделювання та симуляції гетерогенних мереж користувачі хотіли б мати модульні програмні засоби, але з різноманітними масштабами моделей. Одні хочуть мікроскопічних моделей, інші віддають перевагу мезоскопічним моделям, а ще інші хочуть деякі мультимасштабні моделі. Якщо йдеться про гетерогенний трафік, моторизований та немоторизований, та їх взаємодію, є потреба в простому або автоматизованому методу калібрування. Якщо кількість однопрогонних

транспортних засобів перевищує п'ять відсотків від загального обсягу трафіку, калібрування параметрів буде ускладненим. Таким чином, симуляція різнорідного трафіку буде невдачею у країнах з розвиненою інфраструктурою.

Виходячі з перелічених трендів, можна зробити наступний висновок:

Результати оцінки показали, що існуючі системи симуляції можуть оцінювати поточну ситуацію на дорогах і передбачати умови руху. Більшість інструментів симуляції призначені для "міських", "міжміських" або "комбінованих" дорожніх мереж і можуть працювати з реальними даними в реальному часі. Жодна система не надає всієї функціональності; ні одна система, здавалося б, не має фокусу на конкретному напрямку застосування. Деякі з цих систем використовують гібридні моделі (мікро+мезо, мікро+макро, мікро+мезо+макро); у деяких з них є обмеження на зв'язки і т.д. Детальна мережева модель є необхідною. Модель мережі на основі даних ГІС покращить послідовність даних і ефективність, що часто не враховується постачальниками програмного забезпечення. Відсутність онлайн-програмних додатків з трафіковою симуляцією, спеціально розроблених для різнорідних дорожніх транспортних мереж в периферійних регіонах. З урахуванням зростаючої продуктивності систем симуляції руху майбутні дослідження можуть бути спрямовані на подальше розвиток цієї функціональності в системах симуляції для забезпечення реальної трафікової інформації в реальному часі та передбачень короткострокового трафіку в різноманітних широких районах (сільських, міських, міжміських) за допомогою даних від елементів автомобілів без фокусування тільки на шосе, великих магістралях та міських агломераціях. Налаштування надає більше можливостей для розвитку майбутніх застосувань, але також перевантажує деяких користувачів.

2 АНАЛІЗ ВИМОГ ДО МОДЕЛЮВАННЯ РУХУ

Для того, щоб комп'ютери, програми та веб-сайти виконували ті функції, які ми вимагаємо від них, необхідно, щоб програмне забезпечення було побудовано правильно. Невід'ємною частиною сектору інформаційних технологій, яка вимагає навчання та експертизи, є розробка програмного забезпечення. Функціональні вимоги (FR) містять пояснення про послуги, які програма повинна надавати. Вони описують програмне забезпечення або систему програмного забезпечення. Функція - це ніщо інше, як вхідні дані, поведінка та вихід програмної системи. Можливу мету системи можна визначити за допомогою обчислення, обробки даних, бізнес-процесу, взаємодії з користувачем або будь-якої іншої спеціалізованої функції. У програмному інженерії функціональні вимоги називаються також функціональними специфікаціями. Продукт повинен ефективно працювати та мати впізнавані характеристики. Ви можете створити унікальний продукт за допомогою нефункціональних критеріїв, наприклад унікальний візуальний інтерфейс, як показав приклад програми SUMO (Рисунок 2.1). Розуміючи приклади нефункціональних вимог та їх функціонування в додатку, ви можете створити систему, яка задовольняє потреби кінцевих користувачів.

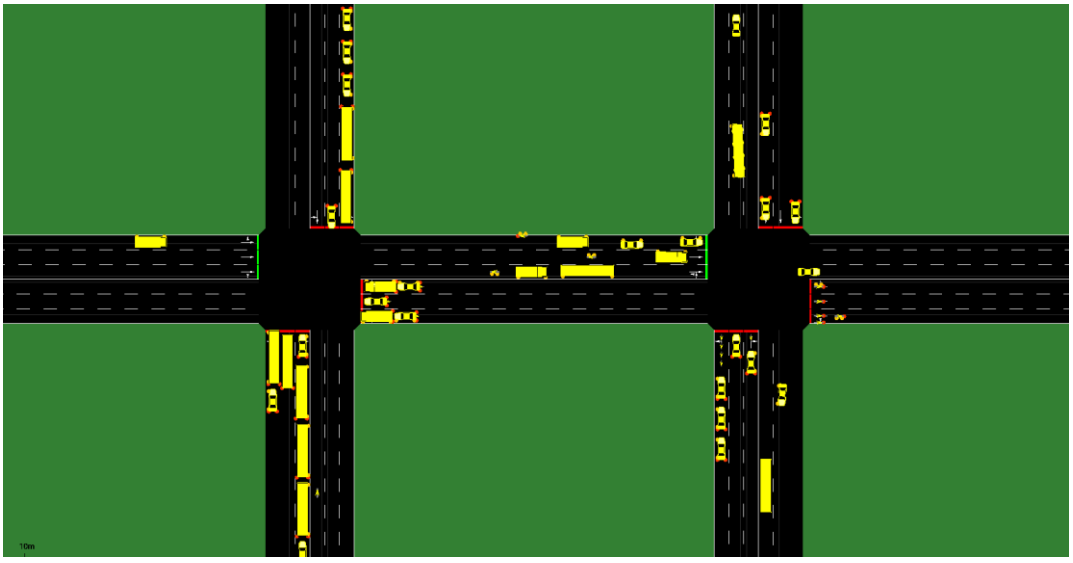


Рисунок 2.1 - Візуальний дизайн програми SUMO

Використання невеликих вибірок даних у передбаченнях є результатом складностей у отриманні початкових даних про трафік, а також обмеженої якості середовища даних. Внаслідок цього багато досліджень використовують складну модель чистої математичної теорії, яка не враховує основних властивостей та еволюційний процес трафіку. Однак надто складні моделі, включаючи нейронні мережі та комбіновані моделі, вимагають витончених обчислень і складних процесів, що робить їх непридатними для використання в реальних короткострокових прогнозах трафіку. Основним обмеженням існуючої системи є людські та природні фактори, такі як проблеми зв'язку, відмови обладнання, серед інших.

2.1 Функціоальні вимоги (ФВ)

Перечень специфікацій, якими повинна бути обладнана система, щоб задовольнити основні потреби кінцевого користувача. Згідно з вимогами контракту, всі ці функції повинні бути вбудовані в систему. Вони показані або описані як вхідні дані, що подаються до системи, операція, що виконується, і призначений результат. На відміну від нефункціональних потреб, це, в основному, критерії, визначені користувачем, які видимі в готовому продукті.

Ось найбільш поширені типи функціональних вимог, які використовуються при розробці системи:

- Бізнес-правила
- Обробка транзакцій
- Вимоги щодо сертифікації
- Адміністративні функції
- Вимоги до звітності
- Рівні авторизації
- Зовнішні інтерфейси
- Відстеження аудиту

- Юридичні та регуляторні вимоги
- Управління історичними даними

Функціональні вимоги системи використовують наступні елементи:

- Інформація про дії на кожному екрані.
- Система повинна мати логіку обробки даних.
- Вона має надавати зведення системних звітів або інших виводів.
- Детальна інформація про робочі процеси, які виконує система.
- Вона повинна чітко вказувати, хто матиме дозвіл на додавання, редагування та видалення даних у системі.
- Функціональний документ повинен надавати інформацію про те, як система буде задовольняти всі відповідні вимоги щодо регулювання та відповідності.

2.2 Нефункціональні вимоги

Якість програмної системи визначається нефункціональними вимогами (НФВ). Вони оцінюють програмну систему за нефункціональними критеріями, такими як відгук, зручність використання, безпека, переносимість та інші критерії, які є важливими для успіху програмної системи. "Як швидко завантажується програма" - це приклад нефункціональної потреби. Системи, які не відповідають нефункціональним критеріям, можуть не здати виконати вимоги користувачів. Ви можете встановити обмеження або обмеження на архітектуру системи через різні зворотні зв'язки Agile, використовуючи нефункціональні вимоги. Наприклад, коли одночасних користувачів більше 1000, програма повинна завантажуватися за 2 секунди.

Опис нефункціональних вимог так само важливий, як і функціональних вимог. Ось деякі з найбільш популярних НФВ:

- Вимоги щодо зручності використання
- Вимоги до обслуговування

- Вимоги до керованості
- Вимоги до відновлення
- Вимоги до безпеки
- Вимоги до цілісності даних
- Вимоги до місткості
- Вимоги до доступності
- Вимоги до масштабованості
- Вимоги до взаємодії
- Вимоги до надійності
- Вимоги до підтримки
- Вимоги до регулювання
- Вимоги до довкілля

Виявлення функціональних вимог дозволяє розробницькій команді підготувати якісне програмне забезпечення. Дослідження дало змогу визначити переваги та недоліки створення типового документу з функціональними вимогами. Вони допомагають визначити, чи надає програма кожну з функцій, вказаних у функціональних вимогах програми. Ви можете визначити функціональність системи або одного з її підсистем за допомогою документа з функціональними вимогами. Виявлення невиконаних вимог допомагає функціональні вимоги та аналіз вимог. Вони допомагають точно визначити бажану поведінку та функціональність системи. Найменш витратні виправлення виявлені під час збору функціональних вимог. Сприяйте користувацьким цілям, завданням або діяльностям.

3 АНАЛІЗ І ВИБІР МЕТОДІВ РЕАЛІЗАЦІЇ ПРОГРАМНОЇ ЧАСТИНИ

3.1 Вибір алгоритмів моделювання руху

Аналіз та вибір методів реалізації програмної частини, а також вибір алгоритмів моделювання руху важливі для створення ефективних та реалістичних симуляційних систем. Ці компоненти забезпечують основу для розробки програм, які могли б відтворювати динамічні сценарії в реальному часі, що є критично важливим для багатьох застосувань, від відеоігор до дослідження трафіку.

На нашому прикладі коду Java, використовуваного для симуляції руху транспорту, можна відзначити декілька ключових аспектів вибору алгоритмів і методів. Перш за все, програма використовує клас **Car** (Рисунок 3.1) для представлення автомобілів.

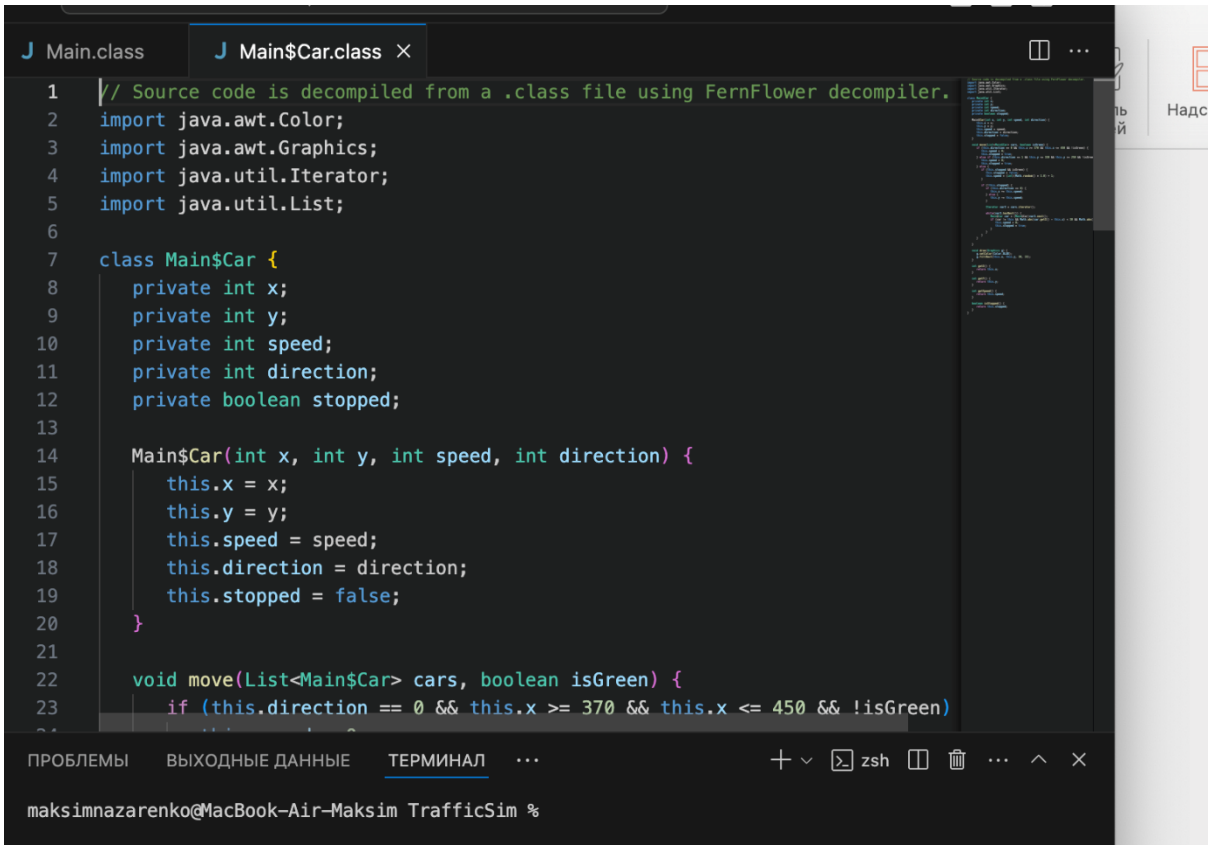


Рисунок 3.1 - Клас Car

Кожен об'єкт **Car** містить атрибути, такі як координати, швидкість та напрямок руху. Це базова модель, яка дозволяє автомобілю переміщуватися в залежності від заданих параметрів. Важливою особливістю є метод **move**, який не тільки змінює позицію автомобіля, але й враховує існування інших автомобілів на дорозі, тим самим уникнення колізій.

Метод колізій у програмі є досить примітивним і базується на перевірці перекриття прямокутників, що представляють автомобілі. Якщо два автомобіля наближаються один до одного на відстань, меншу за певний поріг, їхній рух зупиняється. Це простий, але ефективний спосіб для демонстраційних та навчальних цілей. Такі методи можна удосконалити, використовуючи більш складні фізичні моделі або алгоритми розпізнавання об'єктів, що можуть забезпечити більш реалістичну взаємодію транспортних засобів.

Інший ключовий аспект симуляції — це візуалізація. В програмі використовується клас **TrafficPanel**, який розширює **JPanel** і відповідає за відображення всіх компонентів симуляції, включно з дорогами та автомобілями. Використання стандартних графічних методів AWT для малювання на компоненті дозволяє легко розширювати і модифікувати візуальні аспекти, хоча для більш складних сценаріїв можуть знадобитися спеціалізовані бібліотеки.

Останній значний компонент симуляції — це управління світлофорами. Система використовує булеву змінну для контролю руху, що є прикладом простого, але важливого алгоритму керування процесами у реальному часі. Це підкреслює, наскільки важливою є координація різних аспектів системи для досягнення гладкого та реалістичного моделювання.

Враховуючи це, при розробці симуляційних систем рекомендується звертати увагу не тільки на вибір алгоритмів, але й на їхнє інтегрування з іншими частинами системи для досягнення збалансованої та ефективної моделі.

3.2 Вибір структури даних для представлення доріг та перехрестя

Аналіз та вибір методів реалізації програмної частини є важливими для створення ефективної симуляції дорожнього руху. Одним із ключових рішень при розробці таких симуляцій є вибір структур даних, що використовуються для представлення доріг і перехресть. Наш код демонструє підхід до моделювання доріг та управління трафіком через класи та об'єкти у Java, використовуючи Swing для візуалізації.

В основі симуляції лежить клас *Road* (Рисунок 3.2), що служить контейнером для об'єктів *Car*. Клас *Road* використовує структуру даних типу *List* для зберігання автомобілів, що дозволяє зручно управляти динамічним списком транспортних засобів. Використання списку як основної структури для *Road* є вибором, що забезпечує гнучкість при додаванні чи видаленні автомобілів під час симуляції, а також легке перебирання всіх автомобілів при обчисленні їхнього руху та відображенні на екрані.

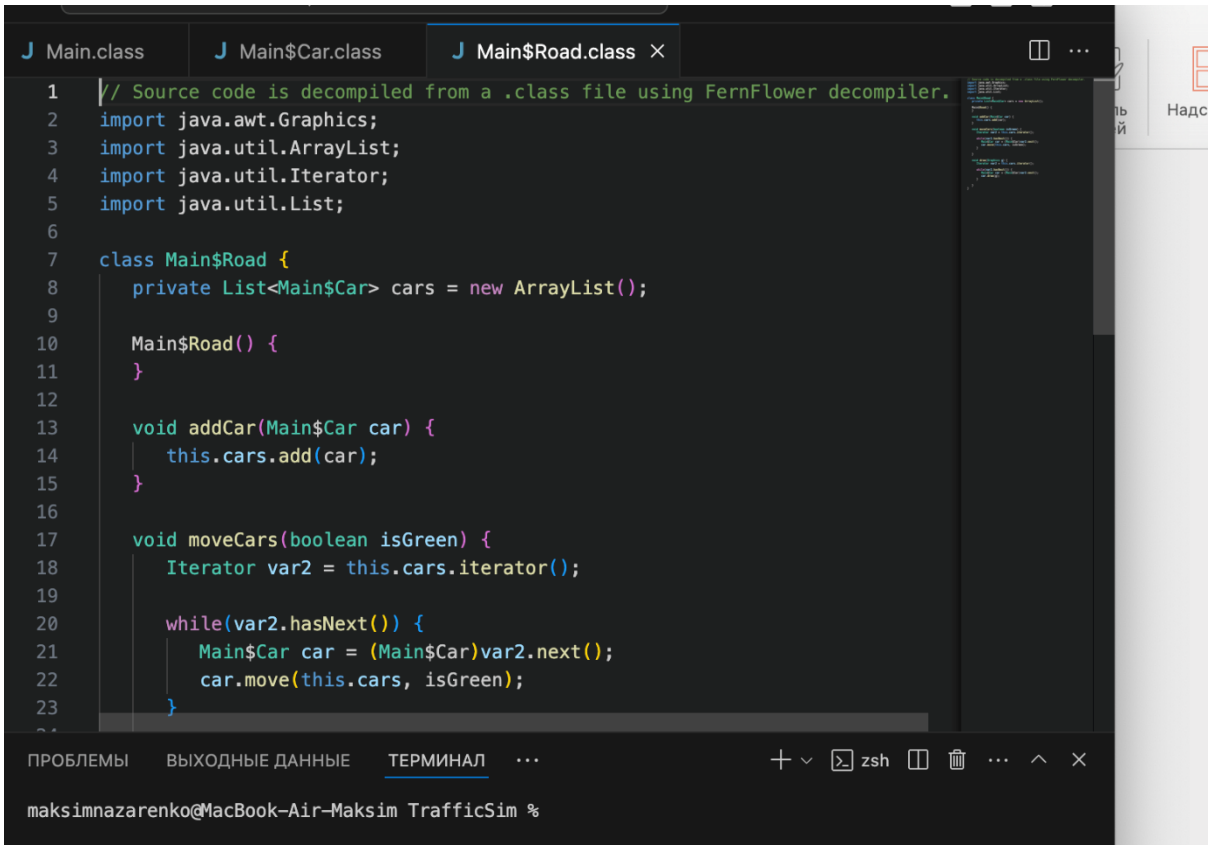


Рисунок 3.2 - Клас Road

Ключовою перевагою такої структури є її динамічність та простота використання, що ідеально підходить для сценаріїв, де кількість елементів (автомобілів) може часто змінюватися. Кожен автомобіль у списку може бути легкодоступний за допомогою ітерації, що є важливим для обробки змін в їхньому стані, таких як швидкість і положення. Однак, такий підхід може не бути найбільш ефективним з точки зору продуктивності при дуже великій кількості автомобілів, оскільки операції пошуку та оновлення стану кожного автомобіля вимагають пропорційно більше часу зі збільшенням розміру списку.

Інший аспект — моделювання перехресть, яке в наведеному прикладі коду не описано детально, але може бути реалізовано шляхом додавання логіки управління світлофорами і правилами руху в клас *Road* або через введення додаткових класів для моделювання перехресть як окремих сутностей. Таке розширення моделі дозволило б управляти складнішими взаємодіями між різними дорогами та напрямками руху, включаючи координацію світлофорів, що могло б значно підвищити реалістичність та ефективність симуляції.

Вибір структур даних і методів їх використання має вирішальне значення для досягнення оптимального балансу між гнучкістю, простотою реалізації, та продуктивністю в програмах симуляції дорожнього руху.

3.3 Вибір алгоритму до візуалізації руху

При аналізі та виборі методів реалізації програмної частини велике значення має вибір алгоритму для візуалізації руху в симуляційних системах. Візуалізація є ключовим елементом у забезпеченні зрозумілості та ефективності інтерфейсу програми, особливо в симуляціях, де користувачі можуть спостерігати та аналізувати складні поведінкові шаблони, такі як дорожній рух. Код Java ілюструє, як можна організувати візуалізацію дорожнього руху використовуючи графічні можливості Java Swing.

Клас **TrafficPanel** (Рисунок 3.3), який є похідним від **JPanel**, використовується для візуального представлення стану дороги та автомобілів на

ній. Цей підхід дозволяє не тільки відобразити елементи ігрового поля, але й динамічно оновлювати їхній стан згідно з логікою програми. В методі **paintComponent,** що перевизначений з класу **JPanel**, виконується візуалізація дороги та автомобілів. Це забезпечується за допомогою викликів графічних методів, таких як **fillRec***t*, для кожного автомобіля та **drawRoad** для доріг. Такий метод дозволяє візуально представити рухомі об'єкти і є широко використовуваним в багатьох типах анімацій та ігор.

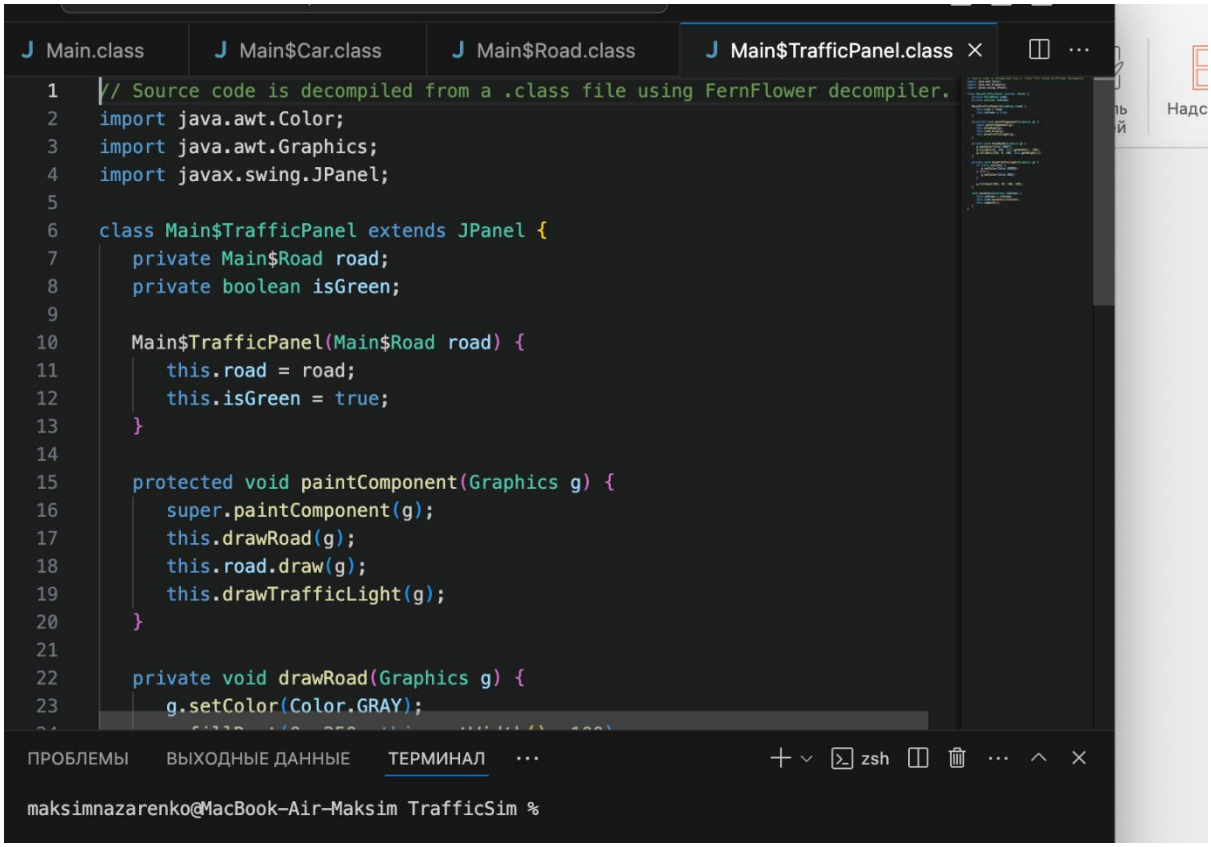


Рисунок 3.3 - Клас TrafficPanel, який використовується для візуальної частини

Вибір Swing як графічної бібліотеки має свої переваги та недоліки. З одного боку, Swing є частиною стандартної Java бібліотеки, що забезпечує хорошу підтримку та легкість використання для створення графічних користувальницьких інтерфейсів. З іншого боку, для більш вимогливих сценаріїв, де необхідна висока продуктивність анімації або використання складних візуальних ефектів, можуть виникнути обмеження через порівняно низьку швидкість виконання та обмежені можливості анімації в Swing. В таких випадках розробники можуть вдаватися до

використання більш спеціалізованих інструментів, таких як JavaFX або навіть інтеграція з OpenGL через бібліотеки третіх сторін.

Під час розробки програмного забезпечення для візуалізації динамічних систем важливо зважати на баланс між доступністю технології, легкістю розробки та вимогами до продуктивності системи. Вибір правильної технології та алгоритму візуалізації може значно вплинути на успіх і прийняття кінцевого продукту користувачами.

3.4 Вибір перехресть та пояснення їх структури

Аналіз та вибір методів реалізації програмної частини важливі для ефективного моделювання та візуалізації транспортних потоків, особливо коли йдеться про перехрестя, які є критичними точками будь-якої дорожньої мережі. В нашому коді для симуляції руху транспорту, структура та логіка роботи перехресть не визначені явно, проте можна розглянути, як можуть бути організовані перехрестя в таких симуляціях.

Перехрестя — це точки, де зустрічаються дві або більше доріг і де виникає взаємодія між різними напрямками руху. В симуляціях, таких як представлена, ключовим аспектом є управління трафіком, що включає регулювання руху та запобігання колізіям. Оскільки в коді використовується лише одна дорога, реальна структура перехресть може бути складнішою та потребувати більш розвиненої логіки для обробки багатонаправленого руху та різних типів транспорту.

Структура перехрестя може бути реалізована за допомогою визначення "гарячих точок", де траєкторії автомобілів перетинаються. Для кожного такого місця можна встановити правила пріоритету, наприклад, за допомогою семафорів або знаків STOP. Автомобілі, що під’їжджають до перехрестя, могли б перевіряти, чи є перехрестя зайнятим іншим транспортом, і відповідно зупинятися або продовжувати рух.

У складних симуляціях перехрестя можуть бути оснащені системами управління трафіком, які адаптуються до поточної дорожньої ситуації. Це може

включати змінні світлофорні цикли, залежні від інтенсивності руху, або навіть інтелектуальні системи, здатні прогнозувати зміни в трафіку на основі зібраних даних та реалізувати проактивні заходи для покращення потоку транспорту.

Вибір та реалізація перехресть у програмі вимагають детального розуміння трафічної логіки та можливостей керування трафіком. Інтеграція ефективних алгоритмів управління може значно підвищити реалізм та корисність симуляції, забезпечуючи не тільки візуальне відтворення руху, але й аналітичні інструменти для планування та оптимізації дорожніх мереж.

4 РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Архітектурний огляд

Розробка програмного забезпечення, зокрема такого, що включає симуляцію дорожнього руху, вимагає глибокого розуміння архітектурних рішень, які забезпечують не тільки функціональність, але й масштабованість, відповідність і легкість утримання. В коді Java для симуляції дорожнього руху використовується кілька ключових класів, що реалізують необхідні компоненти системи: **Car**, **Road**, **TrafficPanel** і **TrafficSimulation** (Рисунок 4.1).

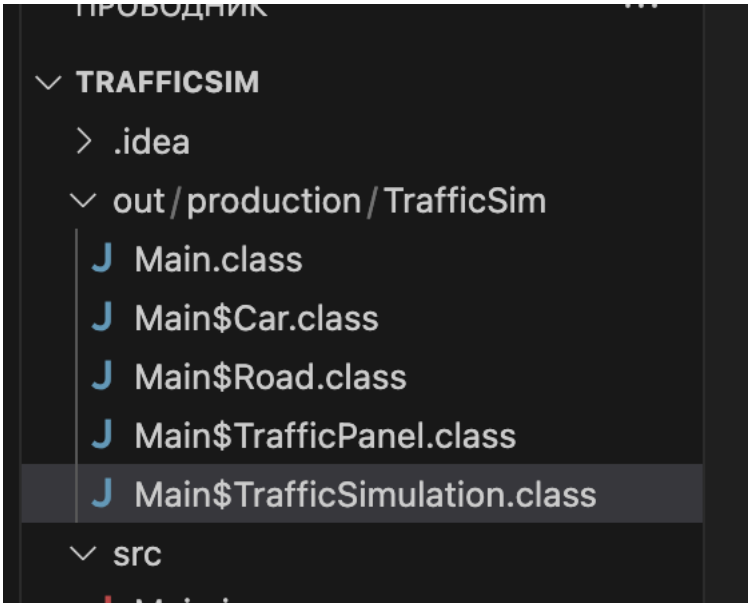


Рисунок 4.1 - Перелік класів

Клас **Car** представляє індивідуальні автомобілі, що мають атрибути для позиціонування, швидкості та напрямку руху. Метод **move** у цьому класі відповідає за рух машини та перевірку на колізії з іншими машинами, що є фундаментальною частиною динамічного симулятора. Це дозволяє системі динамічно реагувати на зміни у середовищі, забезпечуючи реалізм інтеракцій на дорозі.

Клас **Road** є контейнером для множини автомобілів. Цей клас керує колекцією автомобілів і відповідає за їх координований рух. Метод **draw** в **Road** візуалізує всі машини на дорозі, що дозволяє користувачеві візуально оцінити стан

трафіку. Використання класу **ArrayList** для зберігання машин дозволяє легко додавати та видаляти автомобілі, а також ітерувати по списку для обчислення руху та візуалізації.

TrafficPanel є нащадком **JPanel** і відповідає за відображення компонентів симуляції, включаючи дороги та автомобілі. Він використовує перевизначений метод **paintComponent** для відмалювання елементів на екрані, що є центральним для користувацького інтерфейсу програми. Клас **TrafficSimulation**, який є нащадком **JFrame**, управляє основними параметрами вікна програми та ініціює і виконує потоки для симуляції і зміни світлофорів.

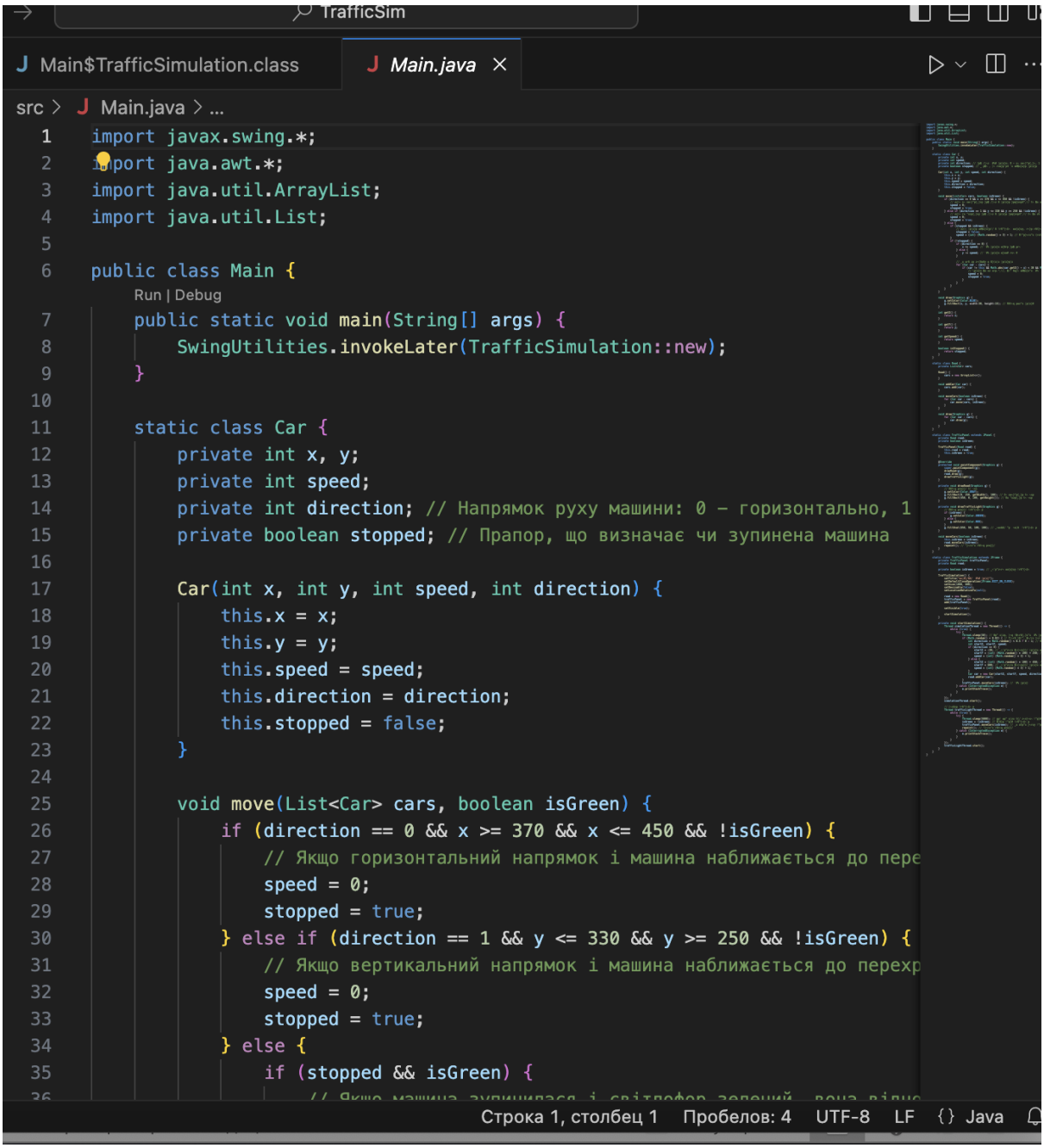
Ця архітектура демонструє використання об'єктно-орієнтованого програмування для розділення відповідальностей між компонентами системи, забезпечуючи високу модулярність та легкість в розширенні та тестуванні. Кожен клас має чітко визначені завдання, що дозволяє розробникам незалежно модифікувати та покращувати окремі частини системи без ризику порушити роботу інших компонентів. В результаті, ця архітектура надає міцну основу для розробки надійних та ефективних програмних рішень у галузі симуляції та візуалізації дорожнього руху.

4.2 Проектування структури

Проектування структури програмного забезпечення вимагає врахування багатьох факторів, включаючи чітке визначення компонентів, їхніх взаємодій та способів їх інтеграції у загальну систему. В даному коді, що симулює дорожній рух, можна побачити чітке розділення відповідальностей між компонентами, що відображає добре продуману архітектуру програми.

Основні компоненти, такі як **Car**, **Road**, **TrafficPanel**, і **TrafficSimulation**, кожен виконує власні завдання, маючи чітко визначені інтерфейси для взаємодії з іншими частинами програми. Клас **Car** відповідає за представлення і поведінку автомобілів на дорозі, управління їхнім рухом і взаємодію з іншими автомобілями.

Road агрегує автомобілі і відповідає за їх колективне управління, включаючи переміщення всіх машин на дорозі (Рисунок 4.2).




```
1   import javax.swing.*;
2   import java.awt.*;
3   import java.util.ArrayList;
4   import java.util.List;
5
6   public class Main {
        Run | Debug
7       public static void main(String[] args) {
8           SwingUtilities.invokeLater(TrafficSimulation::new);
9       }
10
11      static class Car {
12          private int x, y;
13          private int speed;
14          private int direction; // Напрямок руху машини: 0 – горизонтально, 1
15          private boolean stopped; // Прапор, що визначає чи зупинена машина
16
17          Car(int x, int y, int speed, int direction) {
18              this.x = x;
19              this.y = y;
20              this.speed = speed;
21              this.direction = direction;
22              this.stopped = false;
23          }
24
25          void move(List<Car> cars, boolean isGreen) {
26              if (direction == 0 && x >= 370 && x <= 450 && !isGreen) {
27                  // Якщо горизонтальний напрямок і машина наближається до пере
28                  speed = 0;
29                  stopped = true;
30              } else if (direction == 1 && y <= 330 && y >= 250 && !isGreen) {
31                  // Якщо вертикальний напрямок і машина наближається до перехр
32                  speed = 0;
33                  stopped = true;
34              } else {
35                  if (stopped && isGreen) {
```

Рисунок 4.2 – Виконуваний файл

TrafficPanel, що є похідним від **JPanel**, забезпечує візуалізацію компонентів симуляції, зокрема, малювання дороги і автомобілів. Нарешті, **TrafficSimulation** інтегрує всі ці компоненти в одне ціле, керуючи основними параметрами програми, такими як створення вікна і старт основного циклу симуляції.

Цей підхід дозволяє досягти високої згуртованості та низької залежності між компонентами, забезпечуючи легкість розширення та модифікації програми. Наприклад, розширення функціоналу автомобілів або зміна логіки руху може бути виконана без втручання в інші частини системи. Також, це сприяє більш ефективному тестуванню, оскільки кожен компонент може бути перевірений окремо від інших.

Така архітектура також сприяє чіткішому розумінню того, як дані перетікають між різними частинами системи, забезпечуючи краще управління станом програми і зменшення можливості помилок, пов'язаних із неконтрольованим доступом до спільних ресурсів. Відповідно до цього, проектування архітектури програмного забезпечення вимагає стратегічного підходу до розподілу відповідальностей між компонентами, що важливо для створення масштабованих, відновлюваних і легко підтримуваних програмних систем.

4.3 Реалізація основних компонентів програми

У процесі розробки програмного забезпечення для симуляції дорожнього руху, ключове значення має чітке визначення та реалізація основних компонентів програми. Це не тільки спрощує розуміння структури програми, але й забезпечує ефективність та легкість у подальшому розширенні та супроводі. У нашому java коді можна виділити декілька основних компонентів: класи **Car**, **Road**, **TrafficPanel**, та **TrafficSimulation**.

Клас **Car** відповідає за представлення індивідуальних транспортних засобів. Він включає атрибути, такі як координати (**x**, **y**), швидкість (**speed**) та напрямок руху (**direction**). Метод **move** у цьому класі реалізує логіку зміни позиції автомобіля на дорозі та перевірку на можливі колізії з іншими автомобілями, що є критично важливим для симуляції реалістичного руху.

Клас **Road** слугує контейнером для зберігання множини автомобілів і управління їх колективним рухом. Цей клас реалізує методи для додавання

автомобілів на дорогу, переміщення автомобілів і відображення дороги та всіх автомобілів на ній. Така організація дозволяє централізовано управляти всіма автомобілями, що полегшує розробку і підтримку комплексних сценаріїв симуляції.

TrafficPanel — це графічний компонент, який відповідає за візуалізацію стану дороги та автомобілів. Цей клас наслідує від **JPanel** і перевизначає метод **paintComponent** для малювання дороги та автомобілів. Це забезпечує графічне представлення даних, що є необхідним для візуальної верифікації правильності симуляції.

Останній компонент, **TrafficSimulation** (Рисунок 4.3), є основним вікном програми і керує загальним процесом симуляції.

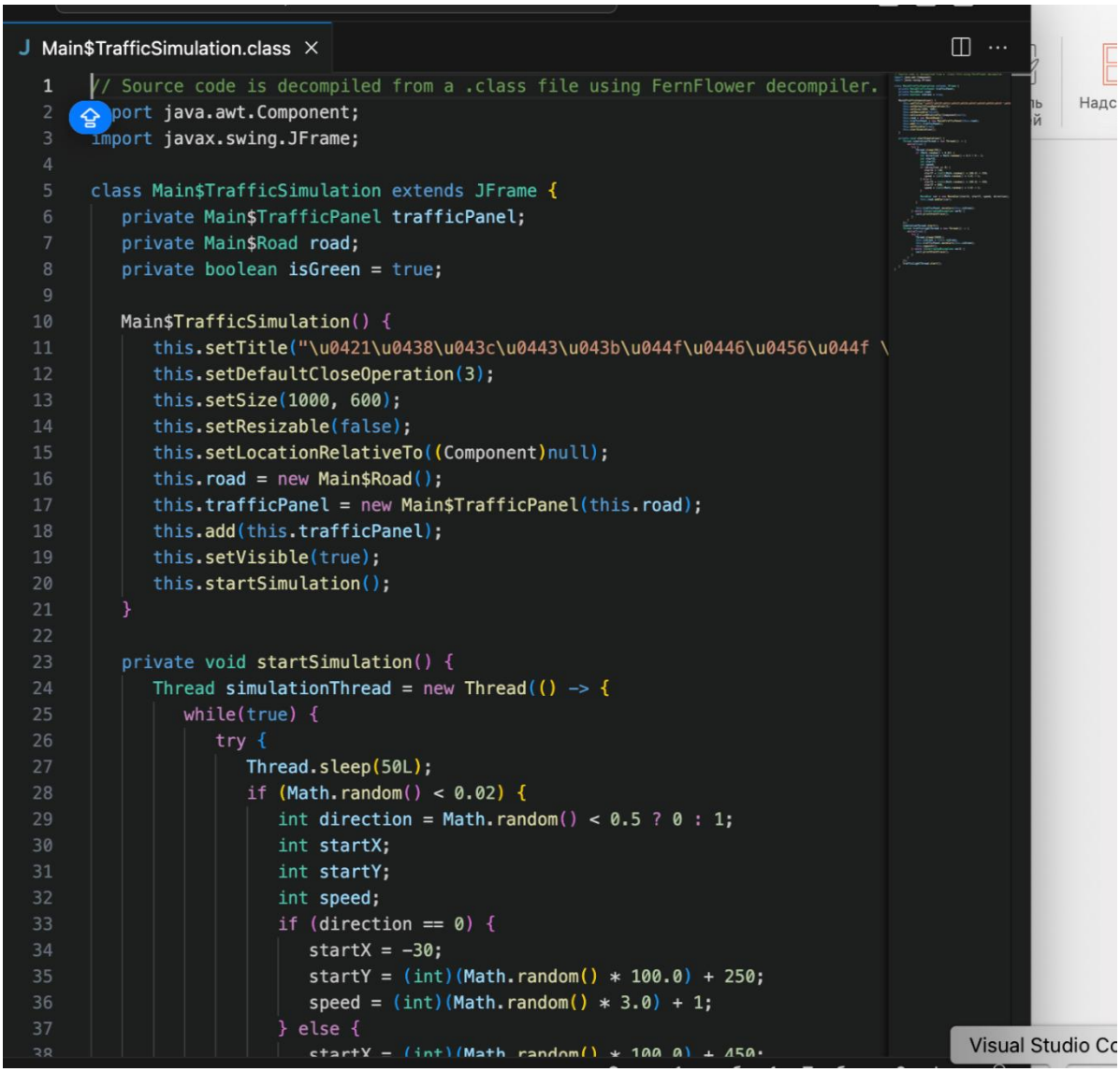


Рисунок 4.3 – Клас TrafficSimulation

Цей клас налаштовує графічний інтерфейс, ініціює потоки для симуляції руху і зміни світлофорів, а також управляє основними циклами програми. Це дозволяє забезпечити координацію всіх частин програми і є ключовим для запуску та зупинки симуляції.

Кожен з цих компонентів відіграє важливу роль у загальній структурі програми, забезпечуючи чітке розділення відповідальності та спрощення управління складними взаємодіями між різними елементами системи.

4.4 Тестування програми на відповідність вимогам

Тестування програми на відповідність вимогам є критичним аспектом розробки програмного забезпечення, особливо коли мова йде про симуляцію динамічних систем, таких як дорожній рух. Це дозволяє переконатися, що всі функції програми працюють правильно та ефективно, а також забезпечує високу якість кінцевого продукту. В прикладі програми для симуляції дорожнього руху, включаючи компоненти, такі як **Car**, **Road**, **TrafficPanel** і **TrafficSimulation**, тестування може включати кілька ключових напрямків.

Перш за все, важливо провести модульне тестування кожного компонента. Наприклад, для класу **Car** можна перевірити коректність розрахунків переміщення та зупинки автомобілів у разі потенційних колізій. Це включає перевірку, що метод **move** правильно оновлює координати **x** та **y** в залежності від напрямку та швидкості руху, а також адекватно реагує на наявність інших машин на дорозі. Тестові сценарії можуть включати створення множини автомобілів і симуляцію їхньої взаємодії на дорозі, перевіряючи, чи правильно програма ідентифікує колізії і зупиняє автомобілі.

Для класу **Road**, що управляє колекцією автомобілів, тестування може зосереджуватися на перевірці ефективності методів додавання та видалення автомобілів, а також правильності їх агрегації та управління загальним рухом. Важливим є також тестування візуалізаційних аспектів у **TrafficPanel**, зокрема правильність малювання компонентів на екрані.

На рівні інтеграційного тестування важливо перевірити, як ці компоненти працюють разом у контексті всієї програми **TrafficSimulation**. Це включає тестування правильності реакції програми на зміни світлофорів і відповідність руху автомобілів заданим правилам. Також варто перевірити, як програма справляється з великим навантаженням, симулюючи велику кількість автомобілів та високу частоту їхніх оновлень.

Завершальним етапом тестування є приймальне тестування, під час якого можна перевірити, чи відповідає програма загальним вимогам та чи є вона готова до використання кінцевими користувачами. Цей етап також може включати відповідність інтерфейсу користувача очікуванням та легкість його використання.

4.5 Аналіз отриманих результатів

Аналіз отриманих результатів після розробки та тестування програмного забезпечення є ключовим етапом, який дозволяє оцінити ефективність програми та ідентифікувати аспекти, що потребують доробки або оптимізації. У випадку розробки симуляції дорожнього руху, програма виконує основні вимоги: забезпечує анімацію руху автомобілів з урахуванням колізій та зміни сигналів світлофора. Таке програмне рішення демонструє візуально зрозумілу модель руху на дорогах, що є важливим для відповідності основним цілям симуляції.

Проте, на основі аналізу виконання програми та відгуків користувачів можуть бути виявлені деякі аспекти, які потребують удосконалення.

По-перше, потрібна більш розширена обробка колізій. В даний час система зупиняє автомобілі при наближенні на малу відстань, що може не відображати реалістичні сценарії поведінки водіїв у реальному світі, де вони можуть зменшувати швидкість, а не повністю зупинятися. Оптимізація логіки руху для введення більш плавних реакцій на дорожні умови може значно покращити реалізм симуляції.

По-друге, програма може бути розширена шляхом введення різноманітних типів дорожніх умов та персоналізації поведінки автомобілів. Це дозволить

симулювати різні дорожні сценарії, наприклад, вплив погодних умов на рух або особливості поведінки водіїв у годину пік. Також може бути корисним введення системи оцінки ефективності дорожнього руху на основі різних метрик, таких як середній час у дорозі або кількість колізій за певний період.

Подальший розвиток програми повинен включати не тільки технічні удосконалення існуючих функцій, але й розширення функціональних можливостей з метою створення більш універсального та адаптивного інструменту для дослідження дорожнього руху. Аналіз отриманих результатів таким чином є відправною точкою для планування майбутніх оновлень та покращень програми.

ВИСНОВКИ

Для цієї бакалаврської роботи, я провів аналіз відомих програм та пакетів ПЗ для моделювання сценаріїв руху на дорогах та порівняв їх сильні та слабкі сторони. Також я створив декілька ітерацій простої програми для візуалізації проїзду через перехрестя зі світлофором. Для створення подібної програми на мові програмування Java, я вивчив основні бібліотеки для візуалізації абстрактного сценарію (тобто Iterator, Color, ArrayList, Component тощо). У майбутньому, я планую ускладнити її через імплементацію інтерфейсу користувача та візуалізацію різних сценаріїв, оскільки ця програма показала себе як непоганий базис для подальшого розвитку.

ПЕРЕЛІК ПОСИЛАНЬ

1. Overview Of Application Of Traffic Simulation Model – [https://www.matec-conferences.org/articles/matecconf/pdf/2018/09/matecconf_mucet2018_03006.pdf]
2. Why Would You Use a Traffic Simulation? – [https://www.ptvgroup.com/en/application-areas/traffic-simulation]
3. Smart-Traffic-Signals-in-India-using-Deep-Reinforcement-Learning-and-Advanced-Computer-Vision – [https://github.com/Ujwal2910/Smart-Traffic-Signals-in-India-using-Deep-Reinforcement-Learning-and-Advanced-Computer-Vision/blob/master/README.md]
4. Trends In Real-Time Traffic Simulation – [https://www.researchgate.net/publication/317421451_Trends_in_Real-time_Traffic_Simulation]
5. Functional and Non Function Requirements of the Traffic Flow Prediction System [https://www.researchgate.net/publication/364910502_Functional_and_Non_Function_Requirements_of_the_Traffic_Flow_Prediction_System]
6. Modeling and Simulation of Vehicular Traffic at Intersection of Roads – [https://www.researchgate.net/publication/353859356_Modeling_and_Simulation_of_Vehicular_Traffic_at_Intersection_of_Roads]
7. A model for the structure of lane-changing decisions – [https://www.sciencedirect.com/science/article/abs/pii/0191261586900123]
8. A STUDY OF TRAFFIC CAPACITY – [https://onlinepubs.trb.org/Onlinepubs/hrbproceedings/14/14P1-023.pdf]
9. Studying Platoon Dispersion Characteristics under Heterogeneous Traffic in India – [https://digitalcommons.unl.edu/cgi/viewcontent.cgi?article=1052&context=civilengfacpub#:~:text=Platoon%20dispersion%20models%20simulate%20the,the%20link]
10. What Is A Point Object? – [https://www.vedantu.com/question-answer/what-is-a-point-object-class-11-physics-cbse-610a152b4d5f1d00a95122de]

ДОДАТОК 1

Лістинг програми:

Main.class

```
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

import java.awt.Color;
import java.awt.Component;
import java.awt.Graphics;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.SwingUtilities;

public class Main {
    public Main() {
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(TrafficSimulation::new);
    }

    static class TrafficSimulation extends JFrame {
        private TrafficPanel trafficPanel;
        private Road road;
        private boolean isGreen = true;

        TrafficSimulation() {
            this.setTitle("Симуляція руху машин");
            this.setDefaultCloseOperation(3);
            this.setSize(1000, 600);
            this.setResizable(false);
            this.setLocationRelativeTo((Component)null);
            this.road = new Road();
            this.trafficPanel = new TrafficPanel(this.road);
            this.add(this.trafficPanel);
            this.setVisible(true);
            this.startSimulation();
        }

        private void startSimulation() {
            Thread simulationThread = new Thread(() -> {
                while(true) {
                    try {
                        Thread.sleep(50L);
                        if (Math.random() < 0.02) {
                            int direction = Math.random() < 0.5 ? 0 : 1;
                            int startX;
                            int startY;
                            int speed;
                            if (direction == 0) {
                                startX = -30;
```

```
                            startY = (int)(Math.random() * 100.0) + 250;
                            speed = (int)(Math.random() * 3.0) + 1;
                        } else {
                            startX = (int)(Math.random() * 100.0) + 450;
                            startY = 600;
                            speed = (int)(Math.random() * 3.0) + 1;
                        }

                        Car car = new Car(startX, startY, speed, direction);
                        this.road.addCar(car);
                    }

                    this.trafficPanel.moveCars(this.isGreen);
                } catch (InterruptedException var6) {
                    var6.printStackTrace();
                }
            }
        });
        simulationThread.start();
        Thread trafficLightThread = new Thread(() -> {
            while(true) {
                try {
                    Thread.sleep(5000L);
                    this.isGreen = !this.isGreen;
                    this.trafficPanel.moveCars(this.isGreen);
                    this.repaint();
                } catch (InterruptedException var2) {
                    var2.printStackTrace();
                }
            }
        });
        trafficLightThread.start();
    }
}

static class TrafficPanel extends JPanel {
    private Road road;
    private boolean isGreen;

    TrafficPanel(Road road) {
        this.road = road;
        this.isGreen = true;
    }

    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        this.drawRoad(g);
        this.road.draw(g);
        this.drawTrafficLight(g);
    }

    private void drawRoad(Graphics g) {
        g.setColor(Color.GRAY);
        g.fillRect(0, 250, this.getWidth(), 100);
        g.fillRect(450, 0, 100, this.getHeight());
    }

    private void drawTrafficLight(Graphics g) {
        if (this.isGreen) {
            g.setColor(Color.GREEN);
        } else {
            g.setColor(Color.RED);
```

```
            }

            g.fillOval(850, 50, 100, 100);
        }

        void moveCars(boolean isGreen) {
            this.isGreen = isGreen;
            this.road.moveCars(isGreen);
            this.repaint();
        }
    }

    static class Road {
        private List<Car> cars = new ArrayList();

        Road() {
        }

        void addCar(Car car) {
            this.cars.add(car);
        }

        void moveCars(boolean isGreen) {
            Iterator var2 = this.cars.iterator();

            while(var2.hasNext()) {
                Car car = (Car)var2.next();
                car.move(this.cars, isGreen);
            }

        }

        void draw(Graphics g) {
            Iterator var2 = this.cars.iterator();

            while(var2.hasNext()) {
                Car car = (Car)var2.next();
                car.draw(g);
            }

        }
    }

    static class Car {
        private int x;
        private int y;
        private int speed;
        private int direction;
        private boolean stopped;

        Car(int x, int y, int speed, int direction) {
            this.x = x;
            this.y = y;
            this.speed = speed;
            this.direction = direction;
            this.stopped = false;
        }

        void move(List<Car> cars, boolean isGreen) {
            if (this.direction == 0 && this.x >= 370 && this.x <= 450 && !isGreen)
{
                this.speed = 0;
                this.stopped = true;
```

```
            } else if (this.direction == 1 && this.y <= 330 && this.y >= 250 &&
!isGreen) {
                this.speed = 0;
                this.stopped = true;
            } else {
                if (this.stopped && isGreen) {
                    this.stopped = false;
                    this.speed = (int)(Math.random() * 3.0) + 1;
                }

                if (!this.stopped) {
                    if (this.direction == 0) {
                        this.x += this.speed;
                    } else {
                        this.y -= this.speed;
                    }

                    Iterator var3 = cars.iterator();

                    while(var3.hasNext()) {
                        Car car = (Car)var3.next();
                        if (car != this && Math.abs(car.getX() - this.x) < 30 &&
Math.abs(car.getY() - this.y) < 15) {
                            this.speed = 0;
                            this.stopped = true;
                        }
                    }
                }
            }

        }

        void draw(Graphics g) {
            g.setColor(Color.BLUE);
            g.fillRect(this.x, this.y, 30, 15);
        }

        int getX() {
            return this.x;
        }

        int getY() {
            return this.y;
        }

        int getSpeed() {
            return this.speed;
        }

        boolean isStopped() {
            return this.stopped;
        }
    }
}
```

Main.java

```
import javax.swing.*;
import java.awt.*;
import java.util.ArrayList;
import java.util.List;

public class Main {
```

```
    public static void main(String[] args) {
        SwingUtilities.invokeLater(TrafficSimulation::new);
    }

    static class Car {
        private int x, y;
        private int speed;
        private int direction; // Напрямок руху машини: 0 - горизонтально, 1 -
вертикально
        private boolean stopped; // Прапор, що визначає чи зупинена машина

        Car(int x, int y, int speed, int direction) {
            this.x = x;
            this.y = y;
            this.speed = speed;
            this.direction = direction;
            this.stopped = false;
        }

        void move(List<Car> cars, boolean isGreen) {
            if (direction == 0 && x >= 370 && x <= 450 && !isGreen) {
                // Якщо горизонтальний напрямок і машина наближається до
перехрестя на червоне світло
                speed = 0;
                stopped = true;
            } else if (direction == 1 && y <= 330 && y >= 250 && !isGreen) {
                // Якщо вертикальний напрямок і машина наближається до перехрестя
на червоне світло
                speed = 0;
                stopped = true;
            } else {
                if (stopped && isGreen) {
                    // Якщо машина зупинилася і світлофор зелений, вона відновлює
рух
                    stopped = false;
                    speed = (int) (Math.random() * 3) + 1; // Встановити швидкість
після зупинки
                }
                if (!stopped) {
                    if (direction == 0) {
                        x += speed; // Рух машини зліва направо
                    } else {
                        y -= speed; // Рух машини знизу вгору
                    }

                    // Перевірка колізій з іншими машинами
                    for (Car car : cars) {
                        if (car != this && Math.abs(car.getX() - x) < 30 &&
Math.abs(car.getY() - y) < 15) {
                            // Машини перекриваються, потрібно зупинити рух
                            speed = 0;
                            stopped = true;
                        }
                    }
                }
            }
        }

        void draw(Graphics g) {
            g.setColor(Color.BLUE);
            g.fillRect(x, y, 30, 15); // Відобразити машину
        }
```

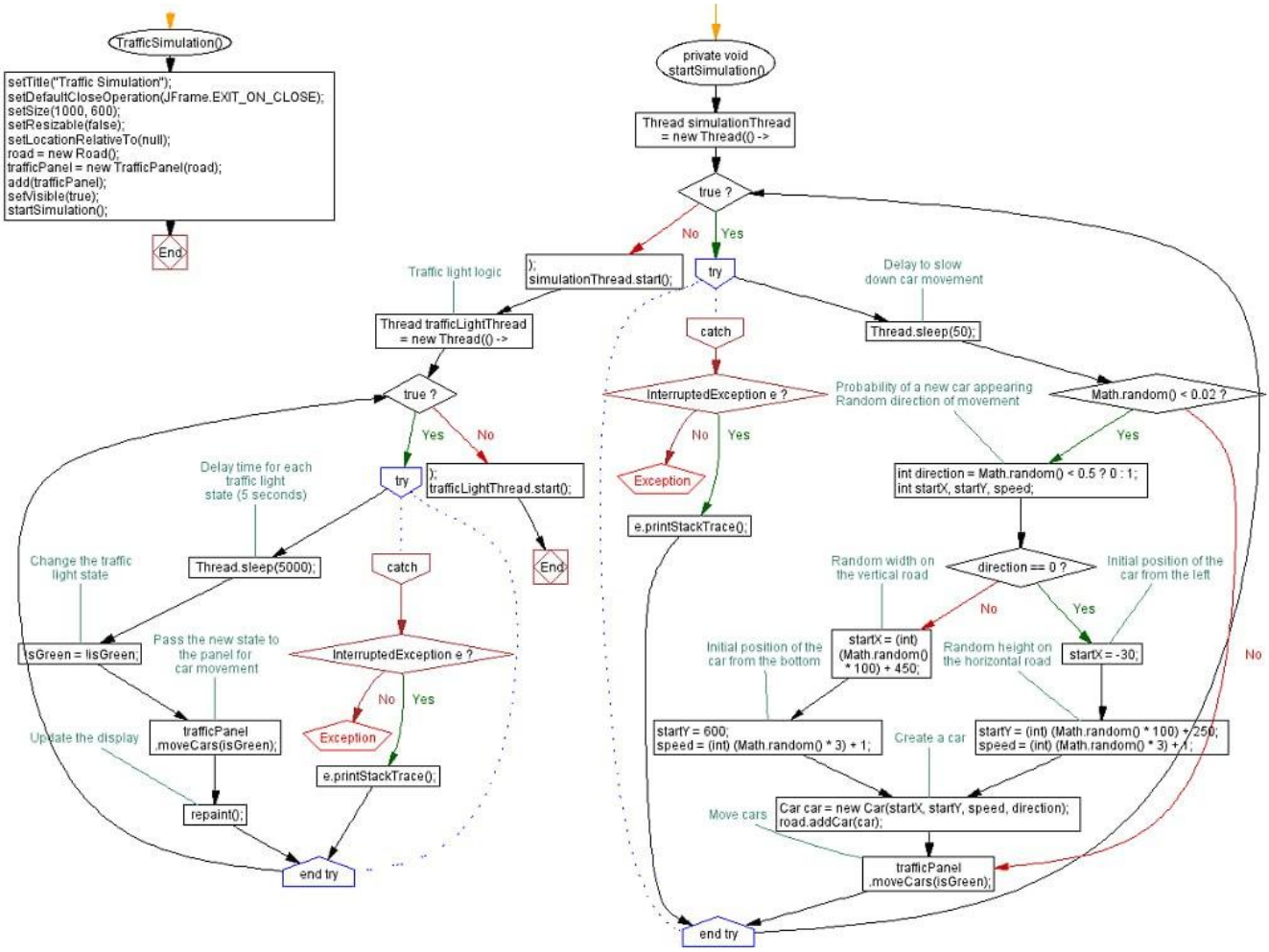
```
    int getX() {
        return x;
    }

    int getY() {
        return y;
    }

    int getSpeed() {
        return speed;
    }

    boolean isStopped() {
        return stopped;
    }
}

static class Road {
    private List<Car> cars;

    Road() {
        cars = new ArrayList<>();
    }

    void addCar(Car car) {
        cars.add(car);
    }

    void moveCars(boolean isGreen) {
        for (Car car : cars) {
            car.move(cars, isGreen);
        }
    }

    void draw(Graphics g) {
        for (Car car : cars) {
            car.draw(g);
        }
    }
}

static class TrafficPanel extends JPanel {
    private Road road;
    private boolean isGreen;

    TrafficPanel(Road road) {
        this.road = road;
        this.isGreen = true;
    }

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        drawRoad(g);
        road.draw(g);
        drawTrafficLight(g);
    }

    private void drawRoad(Graphics g) {
        // Відображення дороги
        g.setColor(Color.GRAY);
        g.fillRect(0, 250, getWidth(), 100); // Горизонтальна дорога
```

```
            g.fillRect(450, 0, 100, getHeight()); // Вертикальна дорога
        }

        private void drawTrafficLight(Graphics g) {
            // Відображення світлофора
            if (isGreen) {
                g.setColor(Color.GREEN);
            } else {
                g.setColor(Color.RED);
            }
            g.fillOval(850, 50, 100, 100); // Позиція та розмір світлофора
        }

        void moveCars(boolean isGreen) {
            this.isGreen = isGreen;
            road.moveCars(isGreen);
            repaint(); // Оновити відображення
        }
    }

    static class TrafficSimulation extends JFrame {
        private TrafficPanel trafficPanel;
        private Road road;

        private boolean isGreen = true; // Початково зелений світлофор

        TrafficSimulation() {
            setTitle("Симуляція руху машин");
            setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
            setSize(1000, 600);
            setResizable(false);
            setLocationRelativeTo(null);

            road = new Road();
            trafficPanel = new TrafficPanel(road);
            add(trafficPanel);

            setVisible(true);

            startSimulation();
        }

        private void startSimulation() {
            Thread simulationThread = new Thread(() -> {
                while (true) {
                    try {
                        Thread.sleep(50); // Затримка, щоб сповільнити рух машин
                        if (Math.random() < 0.02) { // Ймовірність появи нової
машини
                            int direction = Math.random() < 0.5 ? 0 : 1; //
Випадковий напрямок руху
                            int startX, startY, speed;
                            if (direction == 0) {
                                startX = -30; // Початкове положення машини зліва
                                startY = (int) (Math.random() * 100) + 250; //
Випадкова висота на горизонтальній дорозі
                                speed = (int) (Math.random() * 3) + 1;
                            } else {
                                startX = (int) (Math.random() * 100) + 450; //
Випадкова ширина на вертикальній дорозі
                                startY = 600; // Початкове положення машини знизу
                                speed = (int) (Math.random() * 3) + 1;
                            }
```

```
                            Car car = new Car(startX, startY, speed, direction);
// Створення машини
                            road.addCar(car);
                        }
                        trafficPanel.moveCars(isGreen); // Рух машин
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
            simulationThread.start();

            // Логіка світлофора
            Thread trafficLightThread = new Thread(() -> {
                while (true) {
                    try {
                        Thread.sleep(5000); // Час затримки для кожного стану
світлофора (5 секунд)
                        isGreen = !isGreen; // Зміна стану світлофора
                        trafficPanel.moveCars(isGreen); // Передати новий стан
світлофора панелі для руху машин
                        repaint(); // Оновити відображення
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
            });
            trafficLightThread.start();
        }
    }
}
```

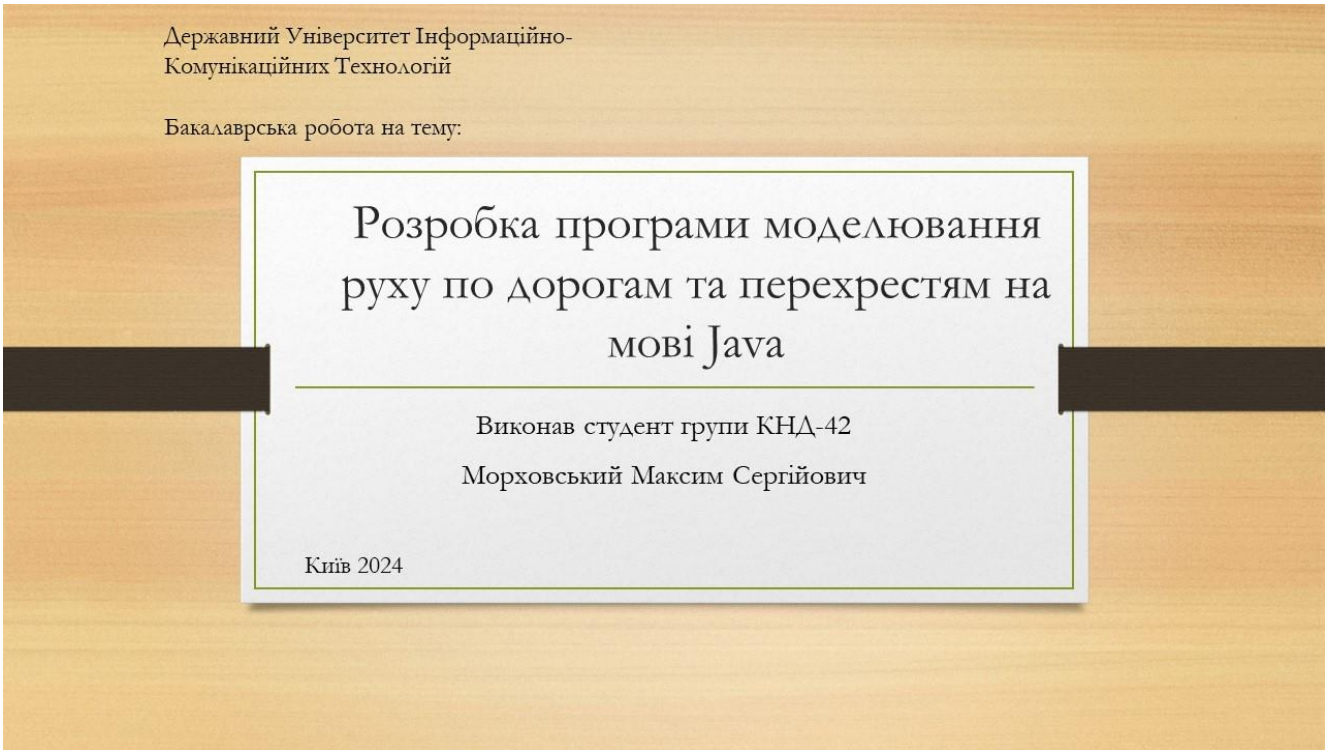
ДОДАТОК 2

Блок-схема програми



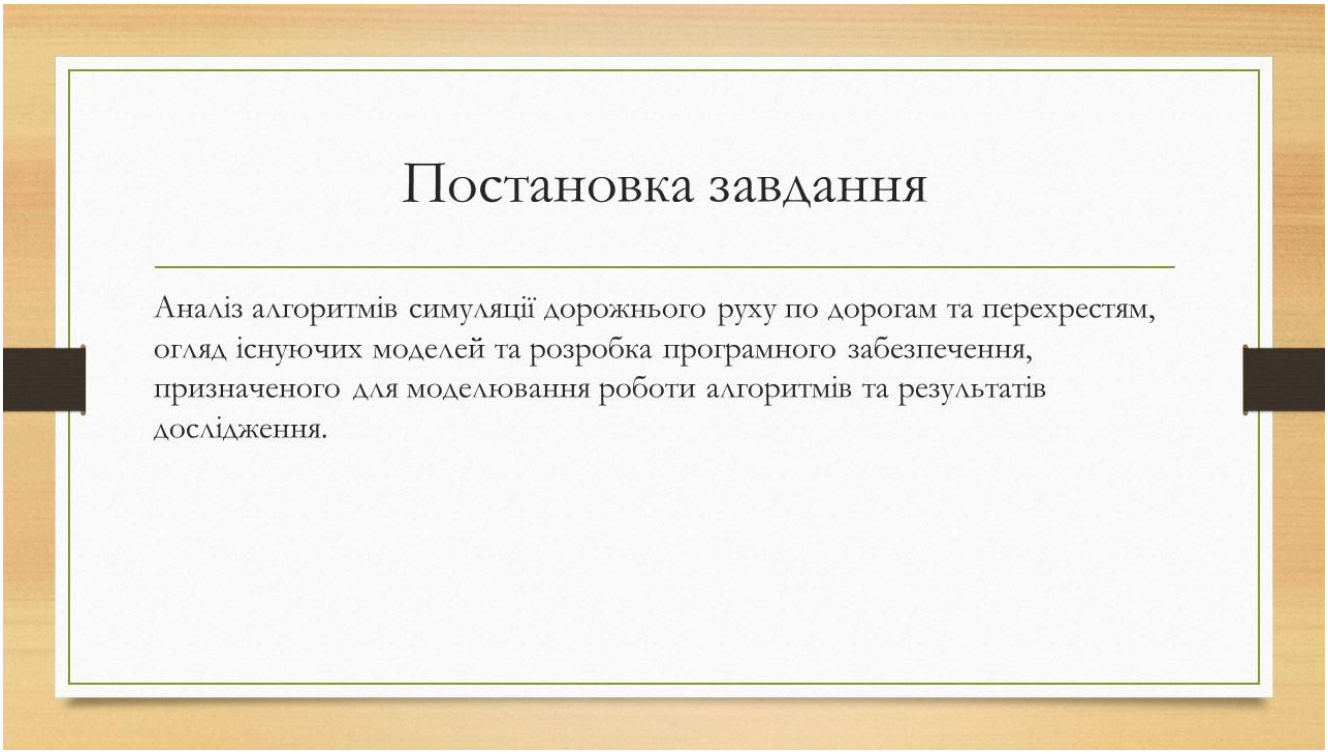
ПРЕЗЕНТАЦІЯ

Слайди з презентації:

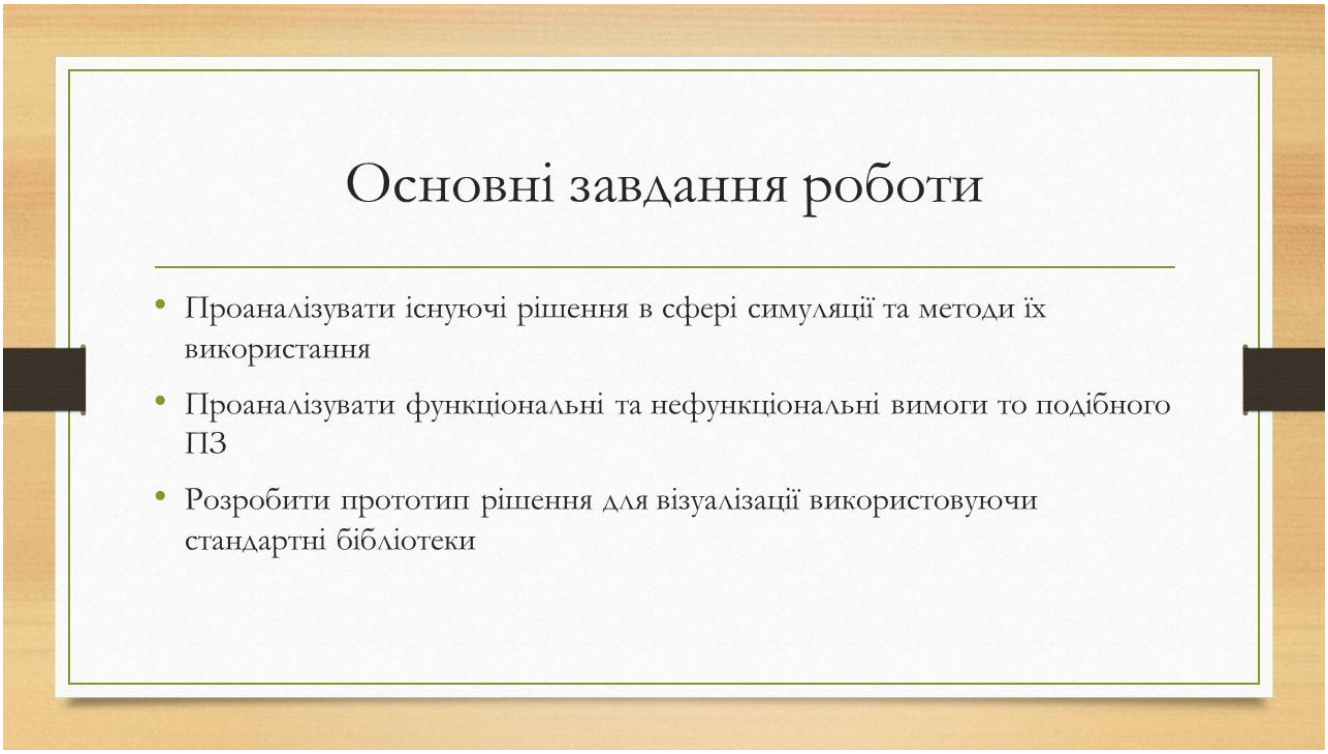


Слайд 1

Слайд 2



Слайд 3



Слайд 4

Актуальність

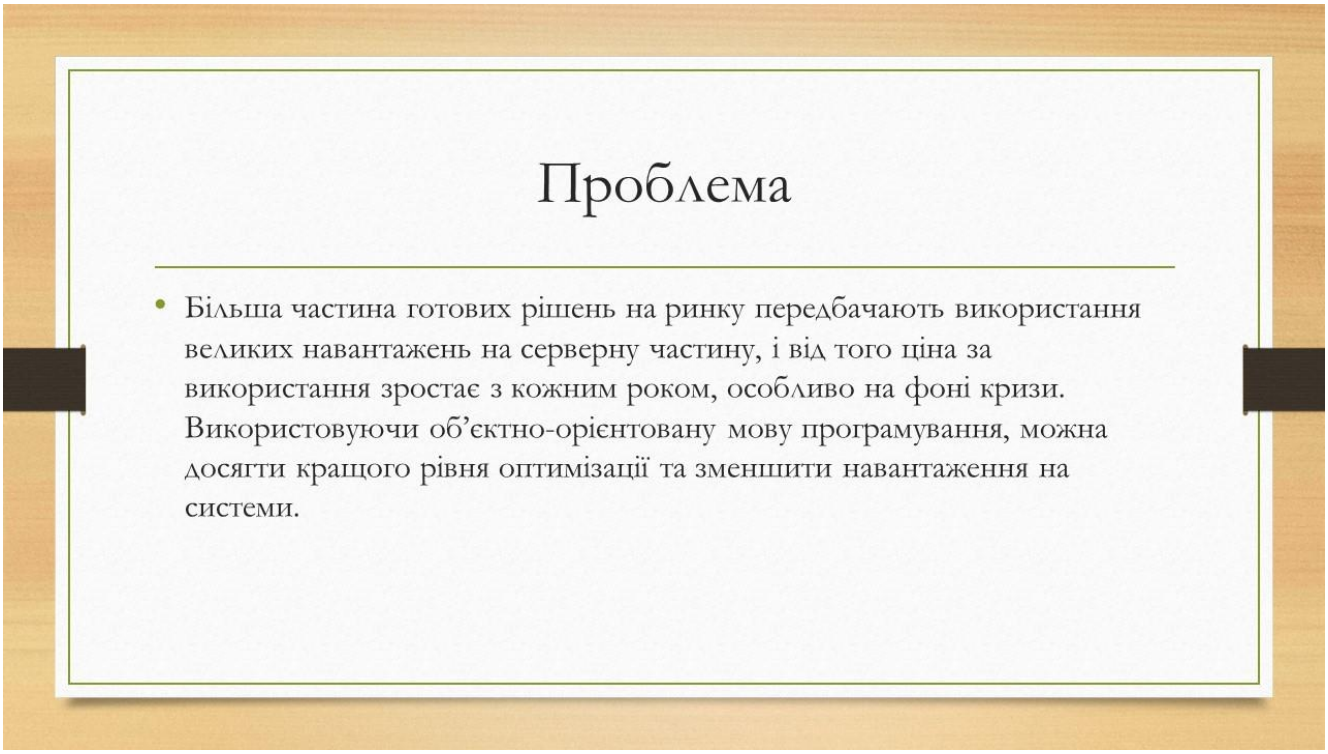
Стимуляційні моделі допомагають зрозуміти вплив різних заходів та обставин на обсяг і потік транспорту за різних обставин. Отже, симуляція руху будує надійну основу для прийняття хороших і економічно ефективних рішень, роблячи рух і мобільність безпечнішими, справедливими та стійкими. Одним словом, це допоможе створити мобільність, орієнтовану на майбутнє.

Слайд 5

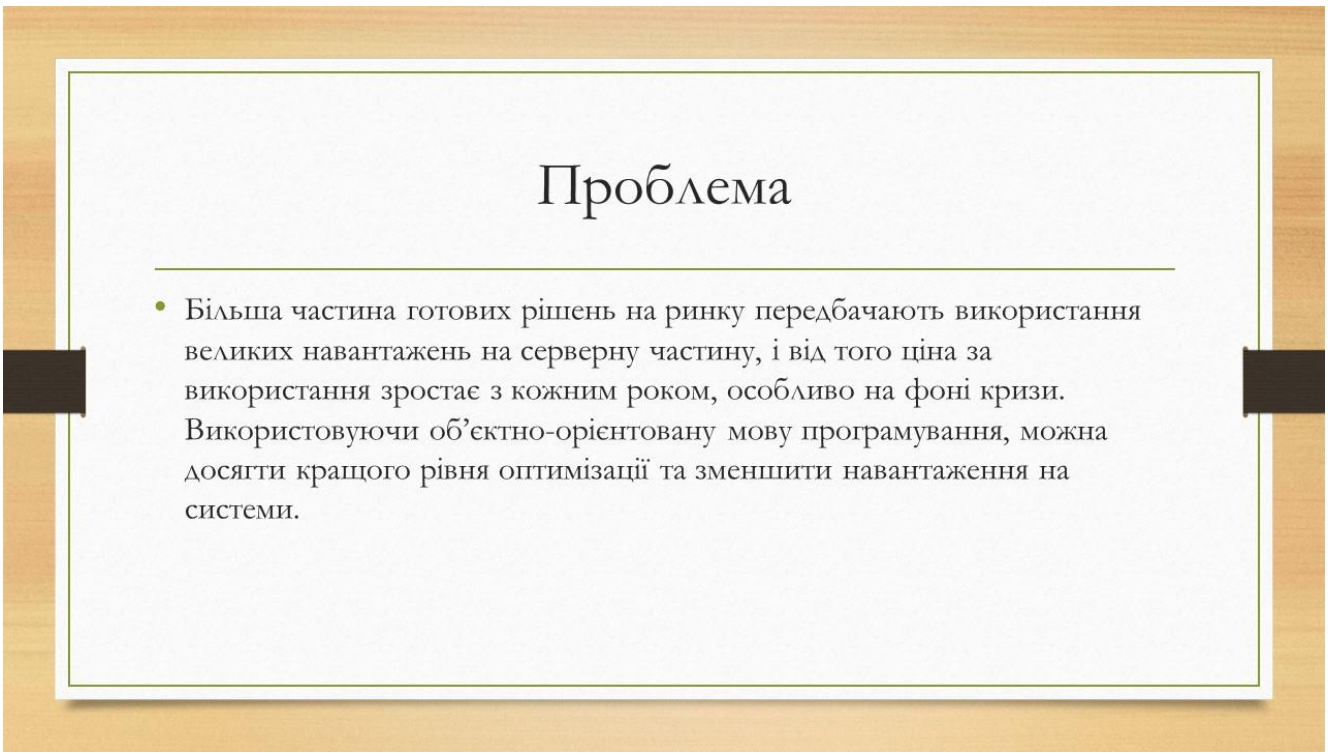
Приклади: табличка з популярними ПЗ для моделювання

Тип моделі	Назва ПЗ	Мета	Враховані параметри	Ким розроблена
Мікроскопічна/ мезоскопічна	Aimsun	Аналіз руху автомобілей, сигналів світлофора та час у дорозі	1) Потік 2) Швидкість 3) Час у дорозі	PTV System Software and Consulting GMBH, Франція
Мікроскопічна	SUMO	Порівняння та оцінка різних алгоритмів пов'язаних з рухом, та оцінка системи світлофорів.	1) Час сигналу світлофора 2) Напрямок повороту 3) Тип перехрестя 4) Суміжні лінії	D.Krajzwwicsz. Institut Fur Verkehrssystem Technik, Берлін, Німеччина
Мікросопічна	Fresim	Вдосконалення геометрії та оперативних можливостей, симуляція геометричних умов	1) Вимір схилів 2) Поведінка водія 3) Знаки 4) Виїзд по схилу	Federal Highway Administration, 1996
Макроскопічна	Saturn	Дослідження впливу вулиць з односторонім рухом, контроль руху на дорогах для автобусів; аналіз та оцінка систем управління трафіком	1) Прийняття проміжків 2) Координація перехресть 3) Швидкість 4) Насиченість потоку	Institute of Transport Studies, University of Leeds, 1979

Слайд 6

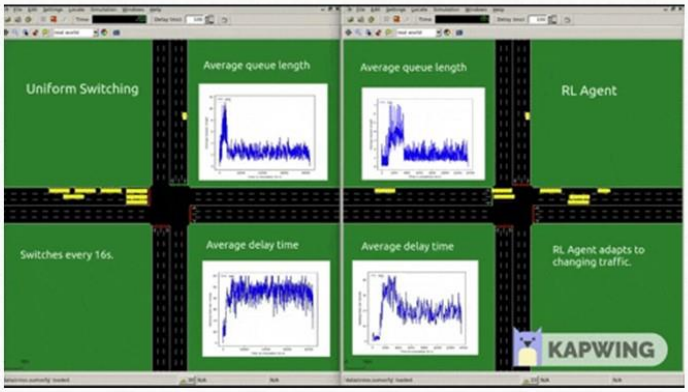


Слайд 7



Слайд 8

Приклади: програма для симуляції руху побудована на основі SUMO

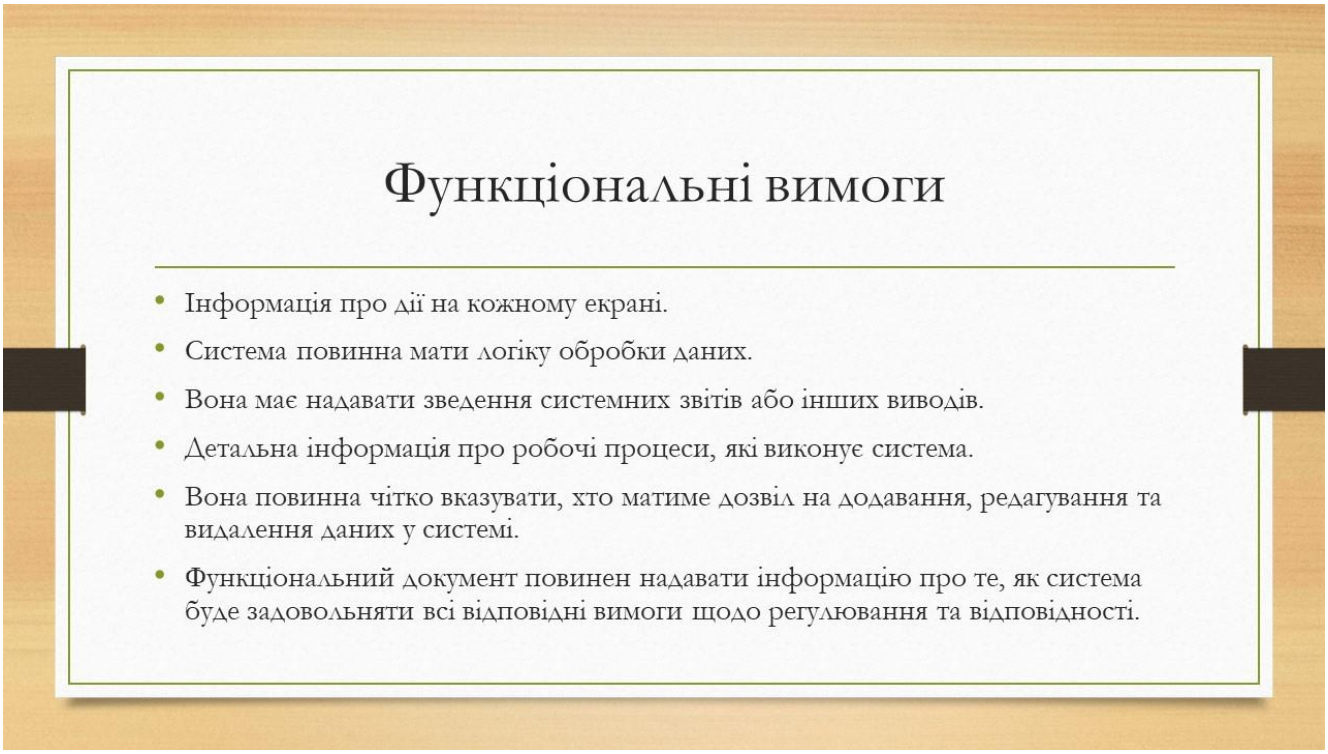


Слайд 9

Концепція запропонованого рішення

Повинна бути створена програма, яка виводить візуальне представлення проїзду перехрестя зі світлофором у окремому вікні, з урахуванням правил дорожнього руху.

Слайд 10


Функціональні вимоги
Інформація про дії на кожному екрані.
Система повинна мати логіку обробки даних.
Вона має надавати зведення системних звітів або інших виводів.
Детальна інформація про робочі процеси, які виконує система.
Вона повинна чітко вказувати, хто матиме дозвіл на додавання, редагування та видалення даних у системі.
Функціональний документ повинен надавати інформацію про те, як система буде задовольняти всі відповідні вимоги щодо регулювання та відповідності.

Слайд 11

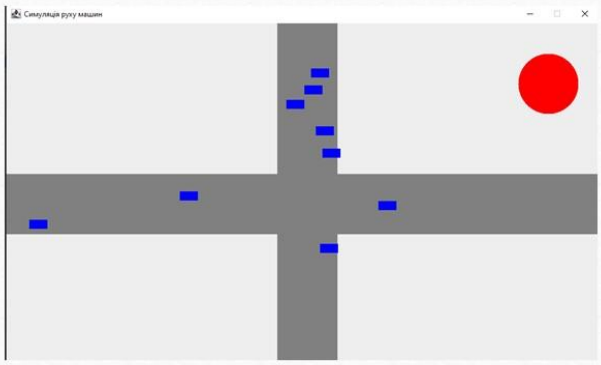
Нефункціональні вимоги

- Зручність використання
- Обслуговування
- Керованість
- Відновлення
- Безпека
- Цілісність даних
- Місткість
- Доступність
- Масштабованість

Слайд 12

Загальний опис додатку

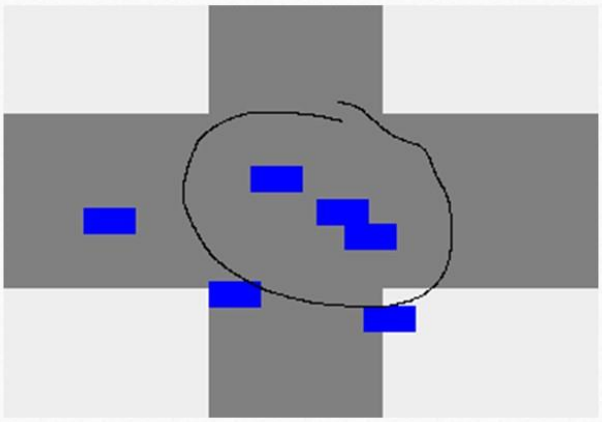
Перш за все, програма використовує клас **Car** для представлення автомобілів. Кожен об'єкт **Car** містить атрибути, такі як координати, швидкість та напрямок руху. Це базова модель, яка дозволяє автомобілю переміщуватися в залежності від заданих параметрів. Важливою особливістю є метод **move**, який не тільки змінює позицію автомобіля, але й враховує існування інших автомобілів на дорозі, тим самим уникнення колізій.



Слайд 13

Архітектура системи

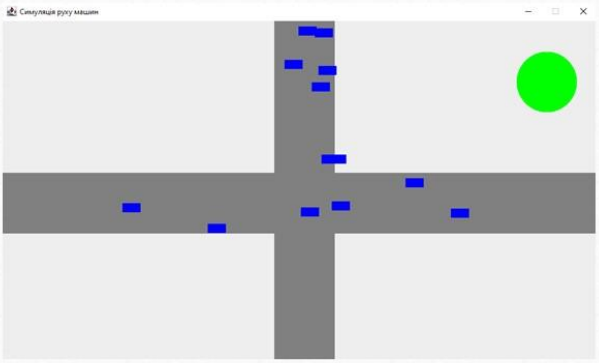
Метод колізій у програмі є досить примітивним і базується на перевірці перекриття прямокутників, що представляють автомобілі. Якщо два автомобіля наближаються один до одного на відстань, меншу за певний поріг, їхній рух зупиняється. Це простий, але ефективний спосіб для демонстраційних та навчальних цілей. Такі методи можна удосконалити, використовуючи більш складні фізичні моделі або алгоритми розпізнавання об'єктів, що можуть забезпечити більш реалістичну взаємодію транспортних засобів.



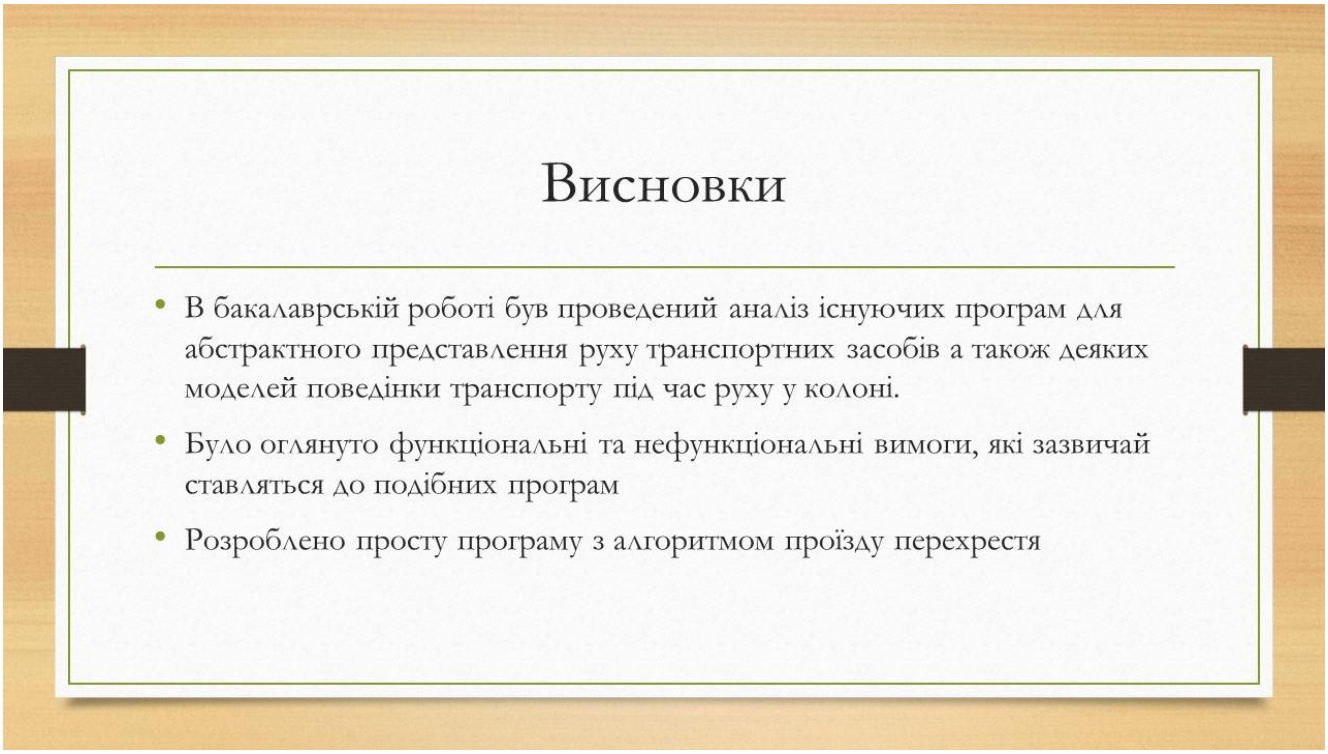
Слайд 14

Результати роботи по розробці

Після кількох ітерацій програми, мені вдалося написати прототип симуляції проїзду з робочим світлофором та урахуванням правил дорожнього руху. Через брак часу викликаний відключеннями світла, я не встиг реалізувати повний потенціал програми, а тому планую дописати її у майбутньому.



Слайд 15


Висновки
• В бакалаврській роботі був проведений аналіз існуючих програм для абстрактного представлення руху транспортних засобів а також деяких моделей поведінки транспорту під час руху у колоні.
• Було оглянуто функціональні та нефункціональні вимоги, які зазвичай ставляться до подібних програм
• Розроблено просту програму з алгоритмом проїзду перехрестя

Слайд 16