| DewMops |
| --- |
| ТЕХНІЧНА ДОКУМЕНТАЦІЯ |
| ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ Mops Simulation |
| Команда DevMops  ІВАНО-ФРАНКІВСЬК |

**Зміст**

[**Технічне завдання** 3](#_gjdgxs)

[**Використані технології:** 4](#_30j0zll)

[**Складові системи:** 4](#_1fob9te)

[**Логічна структура та функціонал** 5](#_3znysh7)

[**Веб-сайт** 5](#_17dp8vu)

**Алгоритмічні складові: імітаційна модель** [6](#_2et92p0)

**Задання транспортної системи 7**

**Ініціалізація суміжних систем**9

**Алгоритм переміщення машин**11

# **Технічне завдання**

Створити імітаційну модель дорожнього руху в місті:

* Створити опис початкових, проміжних та вихідних параметрів моделі.
* Вказати залежності між параметрами у вигляді математичних рівнянь та нерівностей або їх систем.
* Розробити алгоритм розрахунків за моделлю для імітації руху автомобільного транспорту в місті із заданими початковими параметрами.
* Розробити програму, яка за заданими початковими параметрами створюватиме потік даних, які описують рух транспорту в місті.
* Описати структуру вхідного та вихідного файлів.
* Забезпечити введення початкових параметрів з файлу та в діалоговому режимі.
* Забезпечити збереження результатів роботи програми у файл, та відображення їх у вигляді, зручному для перегляду.
* Вказати обмеження та припущення моделі.

Розробити програму візуалізації дорожнього руху у місті

* Створити програму двовимірної візуалізації дорожньої системи міста.
* Візуалізувати рух автомобільного транспорта у місті
* Візуалізувати роботу системи керування автомобільним рухом
* Візуалізувати автомобільний рух у місті згідно з імітаційною моделлю у реальному режимі часу.
* Надати можливість керування швидкістю імітації: пришвидшення, уповільнення, пауза.
* Надати можливість наочного перегляду властивостей об'єктів імітації під час паузи.

# **Використані технології:**

WEB і GUI: HTML5, Css3, JavaScript, Node.Js

PROGRAMME: Python (numpy, osmnx)

# **Складові системи:**

Програмне забезпечення Mops Simulation складається з двох невід’ємних частин: веб-сайт візуалізації та програми-сервера.

Перша частина складається з головної сторінки, сторінки з імітаційною моделлю та сторінки з налаштуваннями.

Друга частина запускається за допомогою консольних команд і керування теж досягається за допомогою маніпуляцій командною строкою.

# **Логічна структура та функціонал**

**Веб-сайт**

На головній сторінці розташоване головне меню, взаємодіючи з якими можна перейти на інші сторінки веб-сайту.

На сторінці з імітаційною моделлю розташоване меню, яке відповідає за керуванням симуляції (запуск, пауза, виключення) та підменю, яке використовується для визначення швидкості симуляції.

На сторінці з налаштуваннями знаходиться панель з налаштуваннями системи: вибір карти транспортної моделі та зміна кількості машин.

## **Алгоритмічні складові**

### **Імітаційна модель**

Алгоритм вирішення задачі створення імітаційної моделі:

1. Задання транспортної системи з файлу .osm або за ім’ям міста за допомогою модулю osmnx;
2. Ініціалізація всіх потрібних систем керування транспортною системою;
3. Переміщення машин по транспортній системі;
4. Запис значень у файли;
5. Повторення циклу.

**Задання транспортної системи**

За допомогою модуля **osmnx** створюємо граф з файла .osm, який можна взяти, для прикладу, з сайту <https://www.openstreetmap.org/> у вкладці “export”, або ж, якщо вибрати місто в налаштуваннях, які доступні на веб-сайті.



Або ж

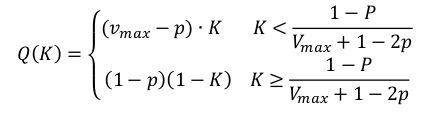


Потім алгоритм перетворює заданий граф у систему, яка є більш зручнішим для маніпуляцій. Використовуючи поділ всієї транспортної моделі на **Nodes**, **Roads**, **Lines**, що відповідно означають - **ноди**, **дороги** і **лінії**. Тобто ноди - це перекрестки та повороти, лінія - це елементарна частинка системи, та дороги - скупчення ліній, які ведуть від певної ноди в іншу ноду. Всі ці елементи мають певні атрубити (Node - type: [“spawn”, “intersect”]; Road - start\_node, end\_node, n\_lines; Line - cells).

Лінія складається з маленьких комірок, наперед заданої довжини, які використовується для переміщення машин по них. Якраз атрибут cells ілюструє список цих комірок.

Приблизний вигляд

Також, алгоритмом передбачено те, що користувач хотітиме побачити якісь кількісні характеристики руху по транспортній системі. Тому була створена функція по їхніх обрахунках по цій формулі:



де Q(K) - означає функцію транспортного потоку за густиною лінії, K - густина (кількість машин на кількість комірок), P - ймовірність того, що машина зменшить свою швидкість на одну одиницю (50 %), vmax - максимальна відносна швидкість (зараз вона становить 3 умовні одиниці).

Ноди мають такі атрибут як type, що визначає тип ноди (“spawn” - в цій ноді будуть створюватися машинки, “intersect” - у цій ноді буде створюватися світлофор). В теперішній версії всі ноди мають однаковий атрибут type = “spawn”.

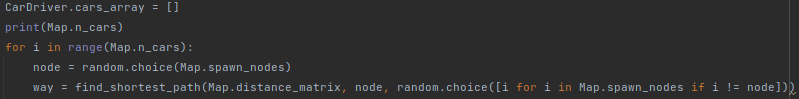
Дороги, як було сказано раніше, складаються з ліній. Якщо дорога двухстороння, то створюються два екземпляра класа Road, щоб задовольнити цю потребу. Атрибути класу: start\_node, end\_node - початкова і кінцева ноди, n\_lines - кількість ліній (в теперішній версій, або однополосна, або двухполосні).

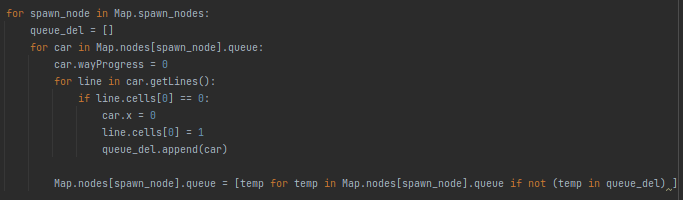
### **Ініціалізація суміжних системи**

Алгоритм ініціалізує клас **CarDriver** та передбачена можливість ініціалізації класу **LightsController**. Що відповідно означають клас для керуваннями машин та світлофорами.



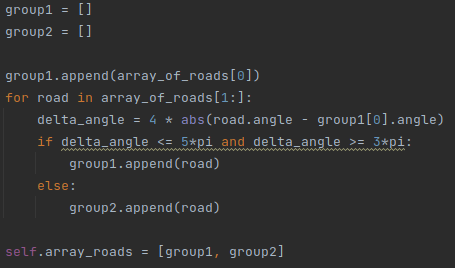
СarDriver має список машин, які будуть рухатися по системі. Спочатку CarDriver ініціалізує всі машини та задає їм найкоротші шляхи за допомогою алгоритма Дейкстри до рандомно взятих нодів.



Після того алгоритм записує машини в черги, щоб не було колізій між ними.

LightsController ініціалізує світлофори, які будуть знаходитися в нодах, які мають в своїх атрибуті **type** значення “intersect”. Вони ініціалізуються так:

1. Поділимо дороги, які зв’язані до певної ноди, на дві підгрупи (визначення по куту дороги);
2. Ініціалізуємо періоди переключення світлофора рандомним чином.

1.

2.

**Алгоритм переміщення машин**

Алгоритм складається з таких етапів:

1. Потрібно спершу машину вставити з черги;
2. Якщо машина на дорозі, то порахувати відстань до найближчої перешкоди;
3. Порахувати відстань, яку машина проїде протягом майбутного часу;
4. Якщо ж майбутня відстань більша ніж довжина лінії, то перевести машину дальше по шляху, або ж коли шлях закінчився, то видалити машину;
5. Алгоритм повторюється.

Також, потрібно відзначити те, що в алгоритм включений елемент рандому, тобто при зміні швидкості є 50% ймовірність зменшення швидкості на одну умовну одиницю.

