DewMops

ТЕХНІЧНА ДОКУМЕНТАЦІЯ

ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ Mops Simulation

Команда DevMops
ІВАНО-ФРАНКІВСЬК

Зміст

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 командним рядком.

Логічна структура та функціонал

Веб-сайт

На головній сторінці розташоване головне меню, взаємодіючи з яким 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.

```
graph = osmnx.graph_from_xml(fileName, simplify=True)
```

Або ж

```
graph = osmnx.graph_from_place(Name, simplify=True)
```

Потім алгоритм перетворює заданий граф у систему, яка є більш зручнішою для маніпуляцій. Використовуючи поділ всієї транспортної моделі на **Nodes**, **Roads**, **Lines**, що відповідно означають - **вузли**, **дороги** і **лінії**. Вузли - це перехрестки та повороти, лінія - це елементарна частинка системи, дороги - скупчення ліній, які ведуть від певного вузла в іншу вузол. 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 кількісні характеристики руху по транспортній системі. Тому була створена функція для їх обрахунків по цій формулі:

$$Q(K) = \begin{cases} (v_{max} - p) \cdot K & K < \dfrac{1-P}{V_{max} + 1 - 2p} \\ (1-p)(1-K) & K \geq \dfrac{1-P}{V_{max} + 1 - 2p} \end{cases}$$

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 %), v_{max} - максимальна відносна швидкість (зараз вона становить 3 умовні одиниці).

Вузли мають такий атрибут як type, що визначає тип вузла (“spawn” - в ньому будуть створюватися машинки, “intersect” - буде створюватися світлофор). В теперішній версії всі вузли мають однаковий атрибут type = “spawn”.

Дороги, як було сказано раніше, складаються з ліній. Якщо дорога двостороння, то створюються два екземпляра класа Road, щоб задовільнити цю потребу. Атрибути класу: start_node, end_node - початковий і кінцевий вузол, n_lines - кількість ліній (в теперішній версії, або однополосна, або двополосні).

Ініціалізація суміжних системи

Алгоритм ініціалізує клас **CarDriver** (клас для керування машинами) та передбачена можливість ініціалізації класу **LightsController** (клас для керування світлофорами).

```
CarDriver.init()
# LightsController.init()
```

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

```
CarDriver.cars_array = []
print(Map.n_cars)
for i in range(Map.n_cars):
    node = random.choice(Map.spawn_nodes)
    way = find_shortest_path(Map.distance_matrix, node, random.choice([i for i in Map.spawn_nodes if i != node]))
```

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

```
for spawn_node in Map.spawn_nodes:
    queue_del = []
    for car in Map.nodes[spawn_node].queue:
        car.wayProgress = 0
        for line in car.getLines():
            if line.cells[0] == 0:
                car.x = 0
                line.cells[0] = 1
                queue_del.append(car)

        Map.nodes[spawn_node].queue = [temp for temp in Map.nodes[spawn_node].queue if not (temp in queue_del)]
```

LightsController ініціалізує світлофори, які будуть знаходитися в вузлах. Вони мають в своєму атрибуті **type** значення “intersect”. Ініціалізуються наступним чином:

1. Поділимо дороги, які прив’язані до певного вузла, на дві підгрупи (визначення по куту дороги);

```
group1 = []
group2 = []

group1.append(array_of_roads[0])
for road in array_of_roads[1:]:
    delta_angle = 4 * abs(road.angle - group1[0].angle)
    if delta_angle <= 5*pi and delta_angle >= 3*pi:
        group1.append(road)
    else:
        group2.append(road)

self.array_roads = [group1, group2]
```

2. Ініціалізуємо періоди переключення світлофора випадковим чином.

```
[randrange(StartPeriodLow, StartPeriodHigh),
 randrange(StartPeriodLow, StartPeriodHigh)],
```

Алгоритм переміщення машин

Алгоритм складається з таких етапів:

1. Вибрати машину з черги;
2. Якщо машина на дорозі, то порахувати відстань до найближчої перешкоди;
3. Порахувати відстань, яку машина проїде протягом майбутнього 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 випадковості, тобто при зміні швидкості є 50 % ймовірність зменшення швидкості на одну умовну одиницю.

```
def CompV(self, gap) -> int:
    self._v = min(self._v + self._a, MaxVelocity)
    self._v = min(self._v, gap - 1)
    if random.randint(0, 10) >= Probability * 10:
        self._v = max(self._v - 1, 0)

    return self._v + self.x
```