# 2. Opis modelu.

## 2.1. Założenia.

Na potrzeby programu została wycięta środkowa część Warszawy. Ulice, które są na mapie to główne drogi w Warszawie. Ma to zastosowanie praktyczne, ponieważ głównymi drogami jeździ się szybciej i najczęściej są one niejednopasmowe.

Pierwszym założeniem są stałe ograniczenia na drogach. W rzeczywistości działają one jak znak drogowy ograniczający prędkość. Ograniczenia stałe losowane są ze zbioru {30km/h, 40km/h,50km/h} – są to realne prędkości na drogach w Warszawie w godzinach szczytu. Oznacza to, że na danej drodze nie można jechać szybciej niż wynosi jej ograniczenie.

Kolejnym założeniem jest losowanie prędkości danej drogi w danej chwili. One też są losowane ze zbioru {30km/h, 40km/h, 50km/h}, ale potem bierzemy MIN(wylosowana wartość, ograniczenie stałe). Istotne jest to, że ograniczenie stałe jest ważniejsze niż chwilowe, czyli jeśli droga ma ograniczenie stałe 40km/h i zostanie wylosowane 50km/h to ta droga będzie miała prędkość 40km/h. Po dojechaniu auta do skrzyżowania, z prawdopodobieństwem 1/3 następuje losowanie, czy prędkości chwilowe dróg się zmieniły. Jeśli prawdopodobieństwo byłoby większa, to wariancja byłaby zbyt duża i trasa przejazdu mogłaby się zmieniać praktycznie co każde skrzyżowanie. W rzeczywistości warunki na drogach zmieniają się w czasie, ale też nie za często.

Niektóre ulice (niezgodnie z rzeczywistością) są jednokierunkowe, co oznacza, że jeśli można przejechać z punktu A do punktu B, to nie można przejechać z B do A. W rzeczywistości mogą to być jakieś roboty drogowe, które wyłączą dany pas ruchu i w efekcie zrobią ulicę jednokierunkową.

Ważnym założeniem są korki, które są deterministyczne (trzeba poprawić). Można zasymulować korek -10km/h na drodze wygenerowanej trasy, co w efekcie w większości przypadków wymusi na programie znalezienie innej drogi.

Na mapie jest 107 skrzyżowań numerowanych od 0 do 106. Program korzysta z algorytmu Dijkstry, z uwzględnieniem odległości skrzyżowań od siebie, prędkości chwilowych i jednokierunkowości.

## 2.2. Algorytm Dijkstry.

1. Wybierz punkt startowy.  
2. Nadaj odległości od punktu startowego S innym wierzchołkom równą nieskończoność.  
3. Utwórz kolejkę priorytetową wszystkich innych wierzchołków grafu niż startowy, gdzie priorytetem jest odległość.  
4. Do póki kolejka nie jest pusta, usuń z kolejki wierzchołek U o najbliższej odległości.  
5. Dla tego wierzchołka U dla wszystkich jego sąsiadów zaktualizuj odległość jeśli   
d(S,U)+d(U,V)<d(S,V).  
6. Jeśli kolejka jest pusta, to z punktu startowego S do wszystkich innych wierzchołków jest wyznaczona najmniejsza odległość.

## 2.3. Schemat krokowy działania programu.

1. Uruchom plik wykonywalny.  
   2. Wybierz punkt startowy i docelowy.  
   3. Wyznacz trasę.  
   4. (Opcjonalnie) Dodaj korki.  
   5. Wykonaj ruch samochodu do skrzyżowania.  
   6. Jeśli samochód nie znalazł się w punkcie docelowym, wróć do punktu 3.  
   7. Jeśli samochód znalazł się w punkcie docelowym, wróć do 2.

(trzeba opisać wyznacz trasę) - rozbudować

o ograniczenia

# 3. Opis struktury programu.

## 3.1. Opis plików i klas.

Program, język, visual, szczegółowy opis

Pliki i klasy programu:

- MainWindow.xaml – plik zawierający interfejs i grafikę programu,

- MainWindow.cs – plik, w którym jest kod wszystkich przycisków,

- Dijkstra.cs – plik, w którym jest klasa Dijkstra,

- Cross.cs – plik , w którym jest klasa cross,

- Warszawa.png (Warszawa2.png, Warszawa3.png, Crosses.png) – plik źródłowy, który w procesie renderowania i obróbki zamienia się w pliki wyżej wymienione.

- Crosses.png – plik wykorzystywany do przedstawiania fragmentu Warszawy.

## 3.2. Opis metod i funkcji.

Metody i funkcje:

- wyznacz Dijkstrę – wyznacza najszybszą trasę od punktu startowego do docelowego,

- pomoc – wyświetla instrukcje obsługi,

- wyznacz linię – rysuje krzywą łamaną od punktu startowego do docelowego,

- ruch pojazdu – rusza samochodem do skrzyżowania,

- LPM – kliknięcie na skrzyżowanie wyświetla jego numer i zaznacza jego sąsiadów,

- PPM – kliknięcie na linię krzywej łamanej generuje korek między dwoma skrzyżowaniami.

## 3.3. Schemat blokowy

## 

## 3.3 interfejs.

## 3.4. Wybrane fragmenty kodu.

# 4. Instrukcja obsługi.

# 5. Bibliografia.

# 6. Zawartość nośnika.

8. Podział pracy. Chreścionko, Litner.

# Oświadczenie.