Sprawozdanie II

Zadanie 6 - Ruch samochodowy

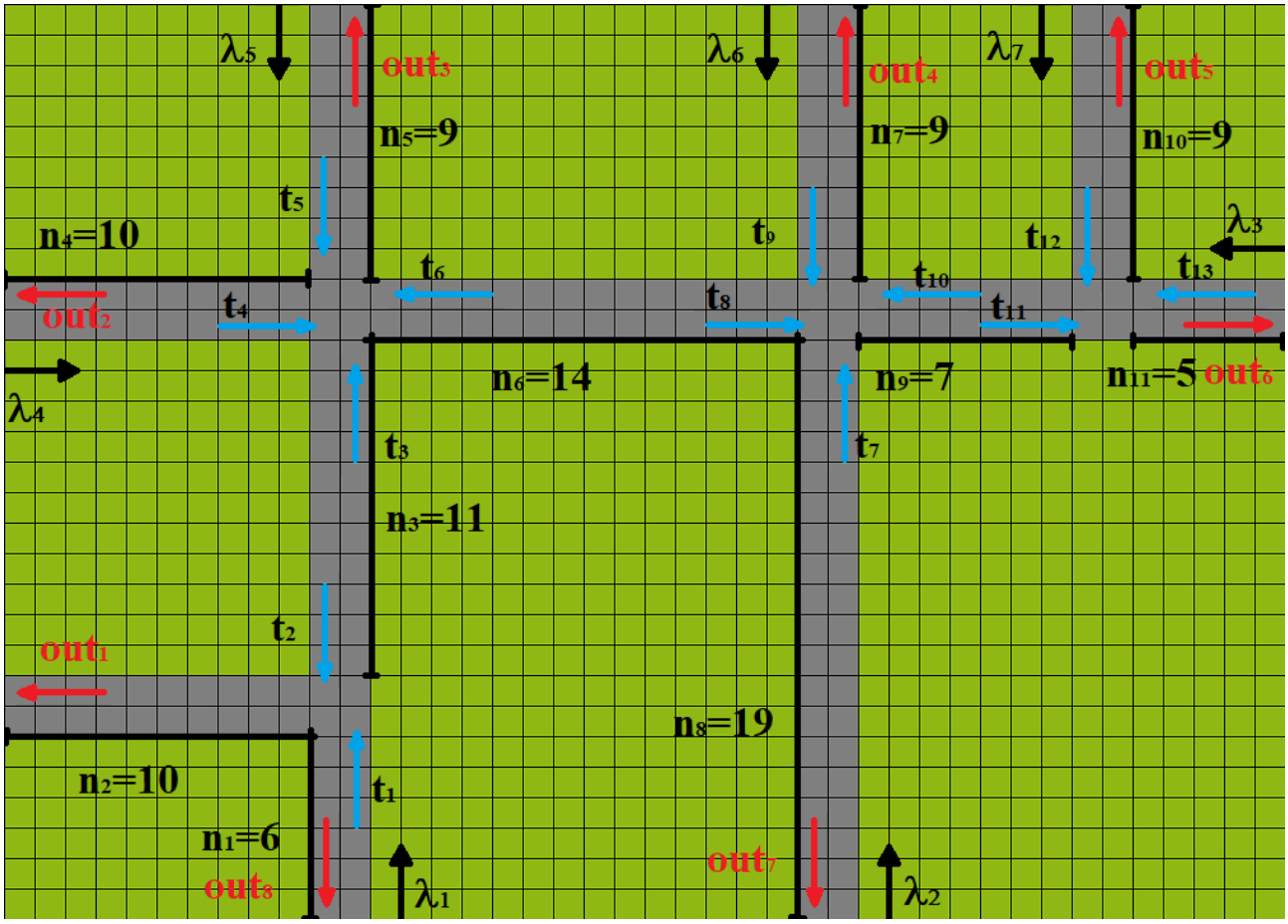
Ada Majchrzak

23 marca 2022

Zasady

- Każdy samochód zajmuje jedną kratkę i porusza się o jedną kratkę, chyba że kolejna kratka jest zajęta lub natrafi na skrzyżowanie.
- Długości ulic w danym kierunku wynoszą $n_1, n_2, \ldots, n_{11}$ ($n_1 = 6, n_1 = 6, n_2 = 10, n_3 = 11, n_4 = 10, n_5 = 9, n_6 = 14, n_7 = 9, n_8 = 18, n_9 = 7, n_{10} = 9, n_{11} = 5$). Dokonałam tutaj niewielkiej modyfikacji - ponieważ moje ulice mają dwa pasy, każdy o szerokości kratki, $n_8 = 19$.
- Samochody na ulicach wjazdowych pojawiają się zgodnie z niejednorodnymi procesami Poissona o funkcjach intensywności kolejno: $\lambda_1(t) = 0.8 + 0.8 \sin\left(\frac{t}{2}\right)$, $\lambda_2(t) = e^{\left(2\sin\left(\frac{t}{12}+3\right)-0.9\right)^3}$, $\lambda_3(t) = 0.5$, $\lambda_4(t) = 0.7 + 0.6 \sin^2\left(\frac{t}{7}+2\right)\cos\left(\frac{t}{3}+1\right)$, $\lambda_5(t) = 0.2 + 0.2 sgn\left(\sin\left(\frac{t}{24}+6\right)\right)$, $\lambda_6(t) = 0.1 + 0.1\sin(\sqrt{t})$, $\lambda_7(t) = 0.3 + 0.3\sin\left(4\sin\left(\frac{t}{2}\right)\right)$.
- W przypadku dotarcia do skrzyżowania S_i, $i = 1, \ldots, 4$ samochód natrafia na sygnalizację świetlną. $t_j = (\tau_j, \pi_j, \sigma_j)$, $j = 1, 2, \ldots, 13$ określa jak długo pali się światło zielone dla samochodów jadących z kierunku t_j (wielkość τ_j), jakie jest prawdopodobieństwo wybrania kierunku na skrzyżowaniu (rozklad π_j) oraz jak długo pali się światło czerwone dla pozostałych kierunków na skrzyżowaniu po zgaśnięciu światła zielonego dla kierunku t_j (wielkość σ_j). Światło zielone zapala się dla kolejnych kierunków zgodnie z ruchem wskazówek zegara. Użytkownik powinien mieć możliwość zmiany parametrów τ_j w trakcie trwania symulacji. Również w przypadku tego podpunktu dokonałam kilku zmian, ale omówię je w dalszej części sprawozdania.
- Samochód na kratce graniczącej ze skrzyżowaniem czeka na zielone światło. Kiedy zielone światło się zapali, samochód wybiera zgodnie z rozkładem π_j (dyskretnym) dalszy kierunek jazdy. Jeśli pierwsza kratka w wybranym kierunku jest wolna, samochód pojawia się na niej, w przeciwnym wypadku czeka na jej zwolnienie.
- Samochody wyjeżdżające z ukladu skrzyżowań znikają i nie bierzemy ich już pod uwagę.
- Oprócz kierunków t_j, $j = 1, 2, \ldots, 13$, dodałam od siebie również kierunki out_k, $k = 1, 2, \ldots, 8$, dla samochodów znajdujących się na drogach wyjazdowych.
- Użytkownik powinien mieć możliwość zapisania animacji w formacie .gif.

Szkic układu



Wstęp

Do stworzenia opisanej wyżej symulacji skorzystałam z biblioteki pygame, dającej niezłe możliwości co do graficznej wizualizacji. Zdaje się, że nie jest to najbardziej optymalne podejście, ponieważ specyfika działania pygame oraz dość skomplikowana natura rozważanego problemu, wymagała ode mnie napisania długiego i dość zawiłego kodu oraz bardzo wnikliwego przestudiowania zachowania układu. W tym sprawozdaniu postaram się jednak opisać moje rozwiązanie tego problemu w jak najbardziej przystępny i zrozumiały sposób.

GUI - oprawa graficzna

Aby czytelnie zwizualizować ruch samochodowy, potrzebowałam przede wszystkim odpowiedniego GUI. Zaczęłam od aspektów typowo estetycznych, które omówię tutaj tylko pobieżnie, gdyż to nie one stanowią istotę zadania:

- Przygotowałam mapę układu oraz grafiki z samochodami i sygnalizacją świetlną.
- Zdefiniowałam klasę przycisku w oknie pygame oraz kilka potrzebnych funkcji (wczytywanie grafik na ekran, pisanie tekstu itd.).
- Stworzyłam menu główne, w którym zawarłam informacje dla użytkownika o tym, jak włączyć nagrywanie symulacji i jakie są jej aktualnie zadane parametry. Umieściłam tam również dwa przyciski - jeden zamykający program, drugi rozpoczynający symulację.
- Zainicjowałam okno pygame dla właściwej symulacji, na którym wczytałam mapę układu (funkcja *play*).
- Aby umożliwić użytkownikowi zapisanie animacji do formatu .gif, w momencie naciśnięcia przycisku V program zaczyna zapisywać kolejne klatki do osobnego folderu. Następnie, przy zakończeniu symulacji, klatki konwertowane są do animowanego gifa i usuwane.

Ruch samochodowy - sygnalizacja świetlna

Sygnalizatory podzieliłam na cztery grupy:

- Grupa 0: Sygnalizatory, na których światło zielone zapala się jako pierwsze.
- Grupa 1: Światło zielone zapala się jako drugie.
- Grupa 2: Światło zielone zapala się jako trzecie.
- Grupa 3: Ostatnie zielone światło.

Każdej grupie przyporządkowałam odpowiednio listę współrzędnych, na których mają wyświetlać się należące do niej sygnalizatory. Następnie zdefiniowałam zmienne def_red, def_yellow, $green0$, $green1$, $green2$ oraz $green3$, których wartości, wprowadzane na początku symulacji przez użytkownika, stanowią parametry układu - to jest właśnie modyfikacja, o której wspominałam na początku sprawozdania. Specyfika biblioteki pygame nie pozwala w prosty sposób zmieniać parametrów w trakcie symulacji, dlatego muszą one zostać zdefiniowane przed jej rozpoczęciem. Ponadto, jak się okazało w trakcie pisania programu, wprowadzanie innych parametrów τ_j oraz σ_j dla każdego kierunku jazdy, ze względu na mój pomysł na kierowanie ruchem w układzie, okazało się niemożliwe, gdyż zanadto komplikowało kod i desynchronizowało działanie całej symulacji. Z tego względu dałam użytkownikowi tylko możliwość zadania czasu palenia się światła czerwonego i żółtego jednakowo dla wszystkich sygnalizatorów oraz czas palenia się światła zielonego dla sygnalizatorów w kolejnych grupach (zmienne $green0$, $green1$, $green2$, $green3$).

Kolejnym krokiem było zdefiniowanie prostej klasy *Signal*, reprezentującej sygnalizatory, funkcji *sign_update*, aktualizującej parametry sygnalizatorów, rekurencyjnej funkcji *repeat*, pozwalającej zachować cykl świateł oraz funkcji *initialize_signals* do inicjowania odpowiednio pogrupowanych obiektów klasy *Signal*. Mając już to wszystko, mogłam do mojej funkcji *play* dodać działające w nieskończoność sygnalizatory.

Ruch samochodowy - pojazdy

Z samochodami zaczęłam bardzo podobnie jak z sygnalizacją. Każdemu wjazdowi λ_i, $i = 1, 2, \ldots, 7$ zadałam początkowe współrzędne pojazdów pojawiających się na mapie. Następnie każdemu kierunkowi jazdy przyporządkowałam (początkowo pustą) listę aktualnie poruszających się samochodów, grupę sygnalizatorów oraz linię zatrzymania przed skrzyżowaniem w przypadku palenia się światła czerwonego. Ponadto, dla ułatwienia kierowania ruchem, podzieliłam kierunki jazdy na cztery grupy: *up* - samochody poruszające się "w górę" mapy, *down* - w dół, $left$ - w lewo, *right* - w prawo. Zdefiniowałam także funkcję *prob* zwracającą (według rozkładu prawdopodobieństwa dla danego kierunku) kolejny kierunek, jaki samochód ma przyjąć wjeżdżając na skrzyżowanie.

Wreszcie najważniejszy element układu - klasa Car, reprezentująca samochody na ulicach. A skoro najważniejszy, to także sprawiający najwięcej problemów. Bardzo skomplikowana w implementacji okazała się funkcja *update*, aktualizująca położenie samochodów na ekranie. Musiałam wziąć pod uwagę nie tylko sygnalizację świetlną, ale także inne pojazdy w układzie czy długości ulic. Zdaje się to proste i logiczne, ale nasz układ zmienia się bardzo dynamicznie i przy każdej jego aktualizacji musimy zmieniać linie zatrzymania, kierunki, zwroty i współrzędne poszczególnych samochodów - stąd właśnie tak wiele instrukcji warunkowych w moim kodzie.

Ruch samochodowy - proces Poissona

Samochody mają pojawiać się na mapie zgodnie z niejednorodnymi procesami Poissona o intensywnościach $\lambda_i(t)$, $i = 1, 2, \ldots, 7$, tzn. według algorytmu:

Niech $M \geqslant max_{[0,T]}\lambda(t)$ (T-horyzont czasowy, na którym generujemy proces), $S_1, S_2, \ldots$ - momenty skoków.

1. Podstaw $t = 0$, $I = 0$.
2. Wygeneruj $U_1 \sim U(0, 1)$.
3. Podstaw $t = t - \frac{1}{M}\log(U_1)$. Jeśli $t > T$, koniec.
4. Generuj $U_2 \sim U(0, 1)$, niezależne od U_1.
5. Jeśli $U_2 \leqslant \frac{\lambda(t)}{M}$, $I = I + 1$, $S_I = t$.
6. Wróć do punktu 2.

W naszym przypadku $T = \infty$ (symulacja ma działać do momentu przerwania jej przez użytkownika), zatem w pierwszej kolejności znalazłam maksima każdej z podanych intensywności dla $t \geqslant 0$:

- $max(\lambda_1(t)) = 1.6$
- $max(\lambda_2(t)) = 3.78483$
- $max(\lambda_3(t)) = 0.5$
- $max(\lambda_4(t)) = 1.3$
- $max(\lambda_5(t)) = 0.4$
- $max(\lambda_6(t)) = 0.2$
- $max(\lambda_7(t)) = 0.6$

W funkcji *play*, przed nieskończoną pętlą *while*, dla każdego wjazdu na mapę zdefiniowałam listę do przechowywania pojazdów niemieszczących się w układzie oraz zadałam początkowy czas rzeczywisty (*time.time*()). Następnie, już w nieskończonej pętli, również osobno dla każdego wjazdu, postępuję zgodnie z algorytmem niejednorodnego procesu Poissona - jeśli spełniony jest punkt 5., dodaję samochód do "poczekalni", a jeśli dodatkowo mamy wolne miejsce na mapie, usuwam samochód z listy i włączam go do ruchu.

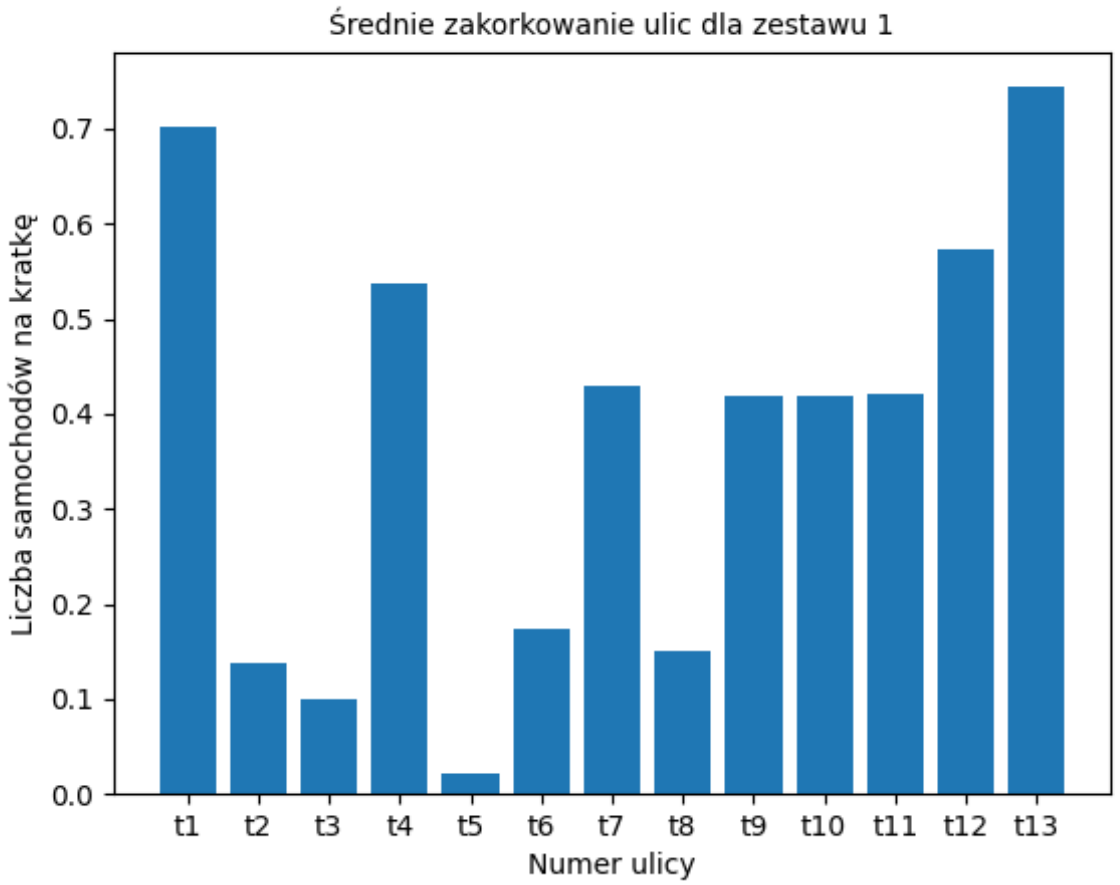
Analiza układu dla trzech zestawów parametrów

W tym sprawozdaniu postanowiłam przyjrzeć się średniemu zakorkowaniu ulic oraz liczbie samochodów opuszczających mapę poszczególnymi wyjazdami. W tym celu wprowadziłam do funkcji *play* argument *num* (domyślnie = 0) oraz flagę logiczną *analysis*. Aby zapisać wyniki analizy, przy wywołaniu symulacji zadajemy argument *num*, czyli numer zestawu parametrów (wykorzystuję go w nazwach plików z wykresami - chciałam, żeby były ponumerowane) oraz ustawiamy flagę na $True$. Analizę przeprowadziłam dla trzech zestawów parametrów, włączając symulację na jedną minutę:

- Zestaw 1: $def\_red = 6$, $def\_yellow = 1$, $green0 = green1 = green2 = green3 = 2$
- Zestaw 2: $def\_red = 8$, $def\_yellow = 1$, $green0 = 1$, $green1 = 3$, $green2 = 3$, $green3 = 2$
- Zestaw 3: $def\_red = 7$, $def\_yellow = 1$, $green0 = 5$, $green1 = 2$, $green2 = 5$, $green3 = 2$

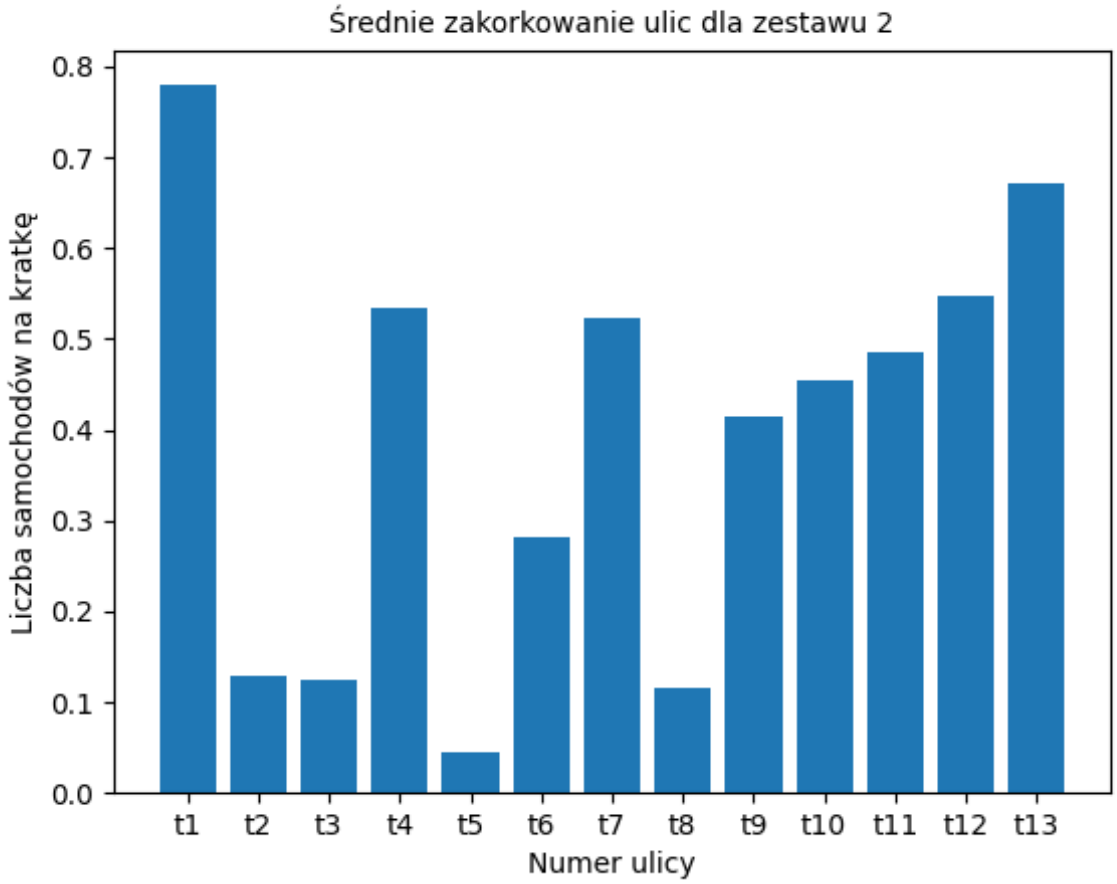
Średnie zakorkowanie

Aby obliczyć średnie zakorkowanie każdej z ulic, po każdej aktualizacji układu obliczam liczbę samochodów na kratkę. Wyniki zapisuję w słowniku, a w momencie zakończenia symulacji wartości słownika nanoszę na wykres słupkowy. Wyniki prezentują się następująco:

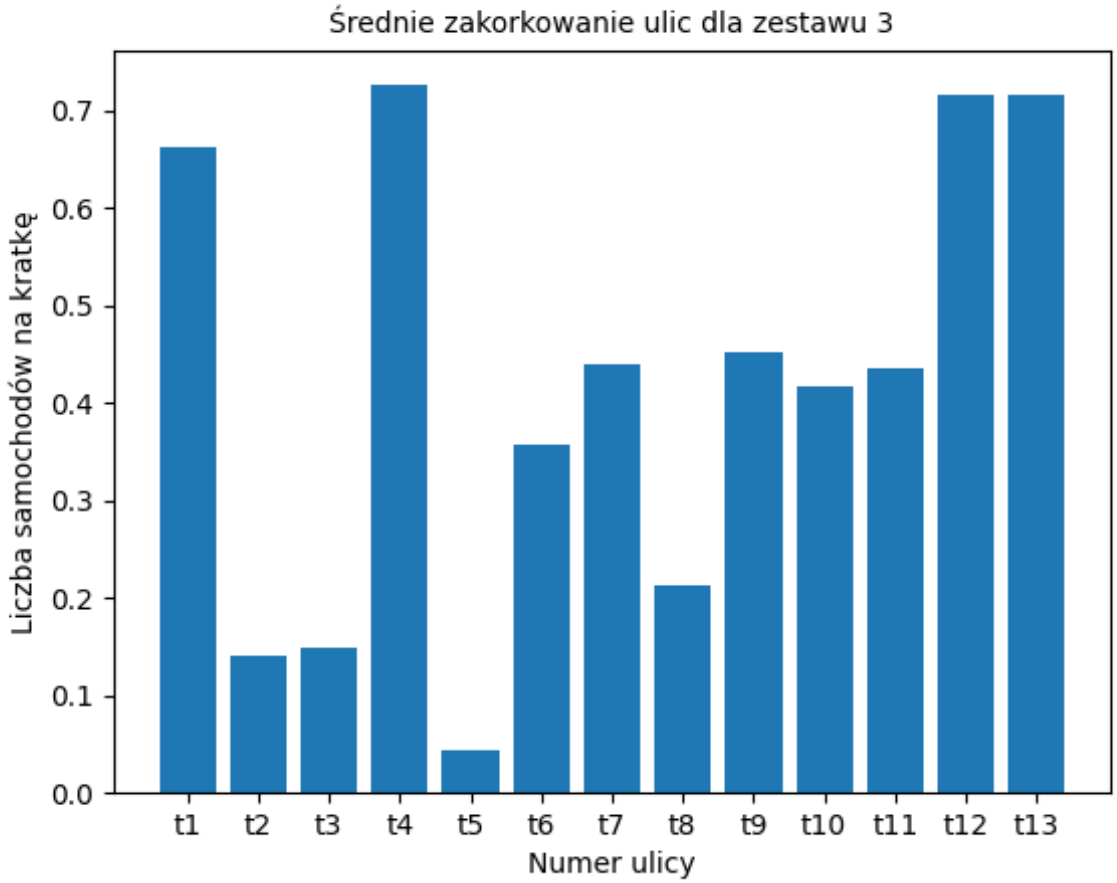


Widzimy z powyższego wykresu, że w przypadku, gdy zielone światło świeci się tak samo długo na każdym sygnalizatorze, największe zakorkowanie obserwujemy dla kierunku t_{13}, podobnie duże dla t_1, a najmniejsze dla kierunku t_5. Możemy domyślać się, że ma to związek z zadanymi dla odpowiednich wjazdów intensywnościami

procesu Poissona - skoro każdy kierunek ma tyle samo czasu na przejechanie skrzyżowania, najwięcej samochodów będzie tam, gdzie najczęściej one wjeżdżają, a więc dla procesu Poissona o najszybszych przyrostach. Zobaczmy jednak, jak zachowuje się uklad dla pozostałych zestawów.



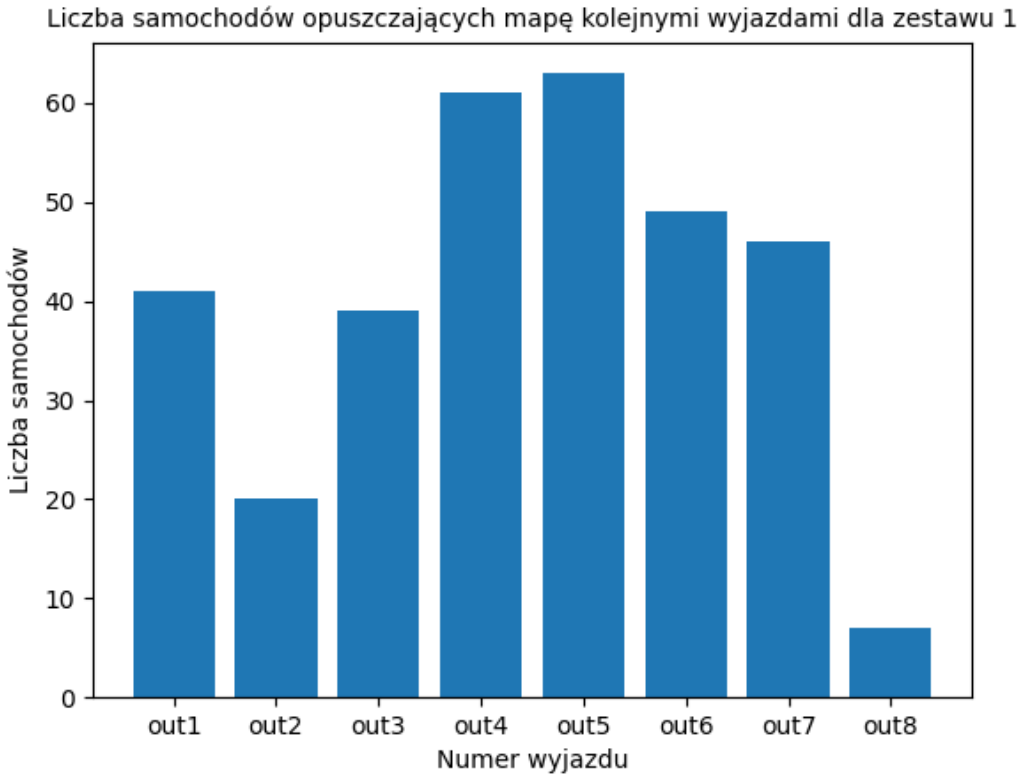
W tym przypadku skróciłam czas palenia się światła zielonego dla kierunków z grupy 0, nieznacznie wydłużyłam dla grup 1 i 2, natomiast dla grupy 3 zostawiłam tak, jak w zestawie 1. Teraz przoduje kierunek t_1, co oczywiście nie jest żadnym zaskoczeniem, biorąc pod uwagę, że znajduje się w grupie, dla której najkrócej pali się światło zielone. Jednak wydłużenie czasu dla kierunku t_{13} dało zaskakująco niewiele - w zestawie 1 jego średnie zakorkowanie wynosiło ponad 0.7 samochodu na kratkę, teraz zaś spadło tylko nieco poniżej tej wartości. Oczywiście może mieć na to wpływ także spora liczba samochodów na ulicy t_{10}, ponieważ to tam z prawdopodobieństwem $\frac{1}{2}$ kierują się pojazdy z t_{13}, ale sprawdźmy jeszcze, co się stanie, jeśli znacznie wydłużymy czas palenia się zielonego światła dla tych najbardziej zakorkowanych kierunków.

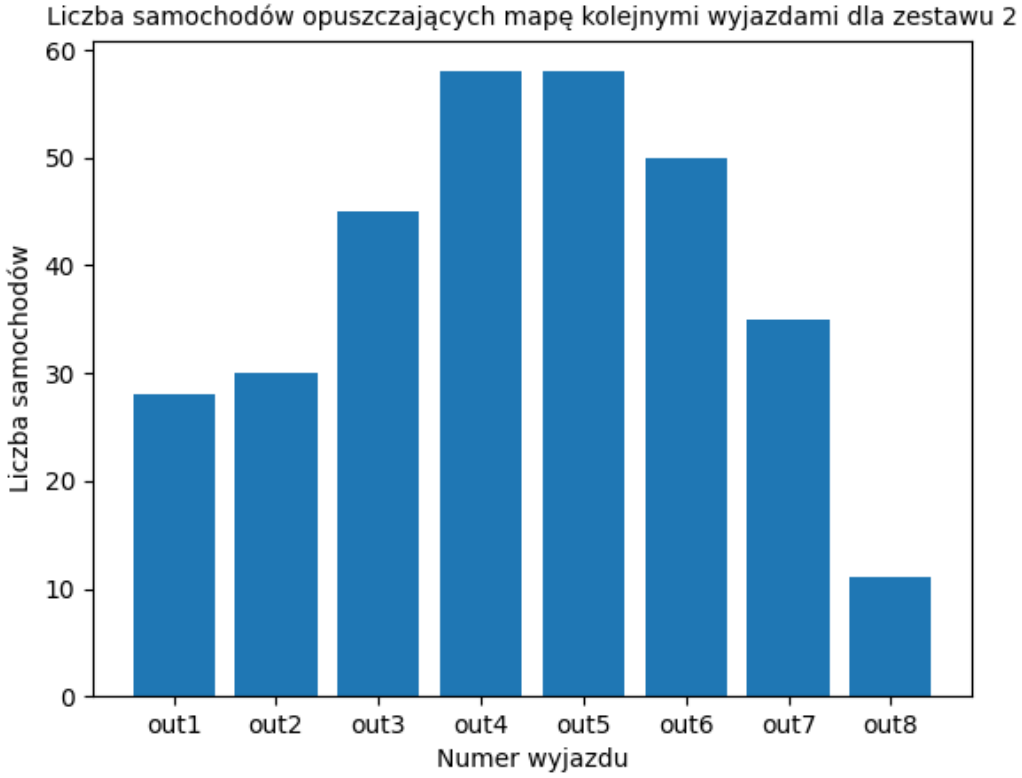


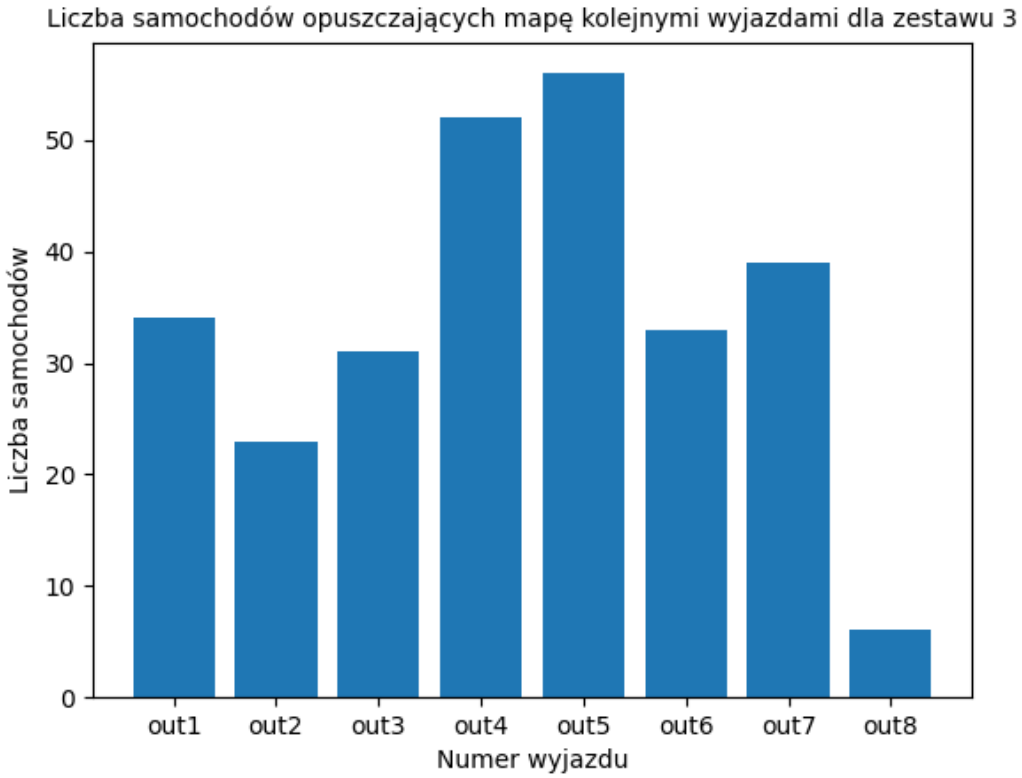
Jak widzimy, w przypadku naszych poprzednich "liderów" znów zmieniło się niewiele. Obserwujemy jednak spore skoki na ulicach t_4 oraz t_{12}. Podejrzewam, że wydłużenie czasu palenia się zielonego światła dla grup 0 i 2 sprawiło, że pojazdy z tych kierunków zaczęły z większą częstotliwością wjeżdżać na kolejne ulice i zajmować więcej kratek, spowalniając tym samym przepływ z pozostałych kierunków na skrzyżowaniach.

Popularność ulic wyjazdowych

Aby sprawdzić, ile samochodów opuszcza mapę poszczególnymi ulicami wyjazdowymi, dokonałam niewielkiej modyfikacji funkcji *update* klasy *Car*. W momencie, kiedy pojazd wyjeżdża z układu (a wiemy, którym wyjazdem), w stworzonym w tym celu słowniku zwiększa się wartość przy kluczu odpowiadającym danemu wyjazdowi. Przyjrzyjmy się wynikom.


Liczba samochodów opuszczających mapę kolejnymi wyjazdami dla zestawu 1
Liczba samochodów
0
10
20
30
40
50
60
out1
out2
out3
out4
out5
out6
out7
out8
Numer wyjazdu


Liczba samochodów opuszczających mapę kolejnymi wyjazdami dla zestawu 2
Liczba samochodów
0
10
20
30
40
50
60
out1
out2
out3
out4
out5
out6
out7
out8
Numer wyjazdu



Tutaj właściwie niewiele się dzieje. Dla każdego zestawu parametrów obserwujemy bardzo zbliżone do siebie wykresy. Samochody najczęściej opuszczają mapę wyjazdami out_4 i out_5, co zapewne wynika ze zdefiniowanych w funkcji *prob* rozkładów prawdopodobieństwa dla poszczególnych kierunków. Dla pozostałych wyjazdów widzimy nieznaczne wahania, ale moim zdaniem jest to wynik losowości symulacji.

Podsumowanie

Tworząc opisaną w powyższym sprawozdaniu symulację, napotkałam wiele problemów. Samochody często zachowywały się w niespodziewany dla mnie sposób, wciąż pojawiały się nowe detale, na które trzeba było zwrócić uwagę. Wymagało to ciągłego szukania innych rozwiązań i obchodzenia pewnych niedoskonałości mojego programu. Tak jak wspomniałam na początku sprawozdania, być może nie wybrałam najprostszego i najbardziej optymalnego podejścia do zadanego problemu, ale efekt końcowy oceniam bardzo dobrze.